الكتاب الأكثر مبيعا في العالم

المرجع
رقم (١)
في العالم

دلفي ٤

Delphi 4 Bible

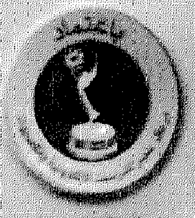
100 %

COMPREHENSIV
AUTHORITATIV
WHAT YOU NEE

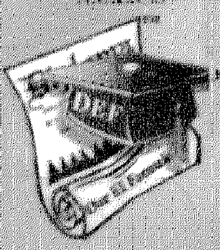
Create graphics, printer, MDI, multimedia, and database programs in short order

Develop user-friendly interfaces with Delphi's Visual Component Library

Integrate custom components into your applications including ActiveX Controls



Translation approved by
N.C.E.C.S





CD-ROM!

تأليف
تـوم سـوان

إعداد
قسم الترجمة بدار الفاروق
د. / خالد العامري






Converted by Tiff Combine - (no stamps are applied by registered version)

دلفـى ٤ بايبـل

Delphi 4 Bible

دار الفاروق للنشر والتوزيع
أكبر مركز فى الشرق الأوسط
لإصدار أحدث الكتب في عالم الكمبيوتر

فرع وسط البلد : ٣ ش منصور المبتديان متفرع من شارع مجلس الشعب
محطة مترو سعد زغلول - القاهرة- مصر
تليفون : ٣٥٥٣٠٣٢ / ٠٢ / ٠٠٢ - ٣٥٤٣٢٠٣ / ٠٢ / ٠٠٢
فاكس : ٣٥٤٣٦٤٣ / ٠٢ / ٠٠٢
فرع الدقى :١٢ ش الدقى - جيزة ت : ٣٣٨١٠٢٢ فاكس : ٨٢٠٧٤

الطبعة العربية الأولى ١٩٩٩
عدد الصفحات : ١٢٢٤ صفحة
رقم الإيداع ٩٩/١١٣٥٧ لسنة ١٩٩٩
الترقيم الدولى 977-307-047-6
الناشر الرئيسى : IDG Books Worldwide. Inc
تاريخ الإصدار للنسخة الإنجليزية الأولى ١٩٩٨

دلفـــى ٤

Delphi 4

بايبل

Bible

تأليف
توم سوان

شكر وتقدير

تتقدم دار:

الفاروق للنشر والتوزيع

بالشكر لقسم الترجمة بالدار وتخص بالذكر كل من :

- د. / خالد العامرى
- م/ محمد شمس — م/جيهان مجاهد

على المجهود الكبير الذى بذل فى ترجمة واعداد وتنفيذ ومراجعة هذا الكتاب .

ABOUT IDG BOOKS WORLDWIDE

Welcome to the world of IDG Books Worldwide.

IDG Books Worldwide, Inc., is a subsidiary of International Data Group, the world's largest publisher of computer-related information and the leading global provider of information services on information technology. IDG was founded more than 25 years ago and now employs more than 8,500 people worldwide. IDG publishes more than 275 computer publications in over 75 countries (see listing below). More than 90 million people read one or more IDG publications each month.

Launched in 1990, IDG Books Worldwide is today the #1 publisher of best-selling computer books in the United States. We are proud to have received eight awards from the Computer Press Association in recognition of editorial excellence and three from Computer Currents' First Annual Readers' Choice Awards. Our best-selling ...For Dummies® series has more than 50 million copies in print with translations in 38 languages. IDG Books Worldwide, through a joint venture with IDG's Hi-Tech Beijing, became the first U.S. publisher to publish a computer book in the People's Republic of China. In record time, IDG Books Worldwide has become the first choice for millions of readers around the world who want to learn how to better manage their businesses.

Our mission is simple: Every one of our books is designed to bring extra value and skill-building instructions to the reader. Our books are written by experts who understand and care about our readers. The knowledge base of our editorial staff comes from years of experience in publishing, education, and journalism — experience we use to produce books for the '90s. In short, we care about books, so we attract the best people. We devote special attention to details such as audience, interior design, use of icons, and illustrations. And because we use an efficient process of authoring, editing, and desktop publishing our books electronically, we can spend more time ensuring superior content and spend less time on the technicalities of making books.

You can count on our commitment to deliver high-quality books at competitive prices on topics you want to read about. At IDG Books Worldwide, we continue in the IDG tradition of delivering quality for more than 25 years. You'll find no better book on a subject than one from IDG Books Worldwide.

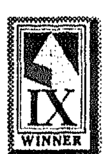


John Kilcullen
CEO
IDG Books Worldwide, Inc.

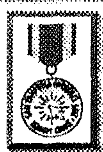
Steven Berkowitz
President and Publisher
IDG Books Worldwide, Inc.



Eighth Annual Computer Press Awards 1992



Ninth Annual Computer Press Awards 1993





Tenth Annual Computer Press Awards 1994



Eleventh Annual Computer Press Awards 1995

IDG Books Worldwide, Inc., is a subsidiary of International Data Group, the world's largest publisher of computer-related information and the leading global provider of information services on information technology. International Data Group publishes over 275 computer publications in over 75 countries. More than 90 million people read one or more International Data Group publications each month. International Data Group's publications include: ARGENTINA: Buyer's Guide, Computerworld Argentina, PC World Argentina; AUSTRALIA: Australian Macworld, Australian PC World, Australian Reseller News, Computerworld, IT Casebook, Network World, Publish, Webmaster; AUSTRIA: Computerwelt Österreich, Networks Austria, PC Tip Austria; BANGLADESH: PC World Bangladesh; BELARUS: PC World Belarus; BELGIUM: Data News; BRAZIL: Anuário de Informática, Computerworld, Connections, Macworld, PC Player, PC World, Publish, Reseller News, Supergamepower; BULGARIA: Computerworld Bulgaria, Network World Bulgaria, PC & MacWorld Bulgaria; CANADA: CIO Canada, Client/Server World, ComputerWorld Canada, InfoWorld Canada, NetworkWorld Canada, WebWorld; CHILE: Computerworld Chile, PC World Chile; COLOMBIA: Computerworld Colombia, PC World Colombia; COSTA RICA: PC World Centro America; THE CZECH AND SLOVAK REPUBLICS: Computerworld Czechoslovakia, Macworld Czech Republic, PC World Czechoslovakia; DENMARK: Communications World Danmark, Computerworld Danmark, Macworld Danmark, PC World Danmark, Techworld Denmark; DOMINICAN REPUBLIC: PC World Republica Dominicana; ECUADOR: PC World Ecuador; EGYPT: Computerworld Middle East, PC World Middle East; EL SALVADOR: PC World Centro America; FINLAND: Mikro PC, Tietoverkko, Tietoviikko; FRANCE: Distributique, Hebdo, Info PC, Le Monde Informatique, Macworld, Reseaux & Telecoms, WebMaster France; GERMANY: Computer Partner, Computerwoche, Computerwoche Extra, Computerwoche FOCUS, Global Online, Macwelt, PC Welt; GREECE: Amiga Computing, GamePro Greece, Multimedia World; GUATEMALA: PC World Centro America; HONDURAS: PC World Centro America; HONG KONG: Computerworld Hong Kong, PC World Hong Kong, Publish in Asia; HUNGARY: ABCD CD-ROM, Computerworld Szamitastechnika, Internetto Online Magazine, PC World Hungary, PC-X Magazin Hungary; ICELAND: Tolvuheimur PC World Island; INDIA: Information Communications World, Information Systems Computerworld, PC World India, Publish in Asia; INDONESIA: InfoKomputer PC World, Komputek Computerworld, Publish in Asia; IRELAND: ComputerScope, PC Live!; ISRAEL: Macworld Israel, People & Computers/Computerworld; ITALY: Computerworld Italia, Macworld Italia, Networking Italia, PC World Italia; JAPAN: DTP World, Macworld Japan, Nikkei Personal Computing, OS/2 World Japan, SunWorld Japan, Windows NT World, Windows World Japan; KENYA: PC World East African; KOREA: Hi-Tech Information, Macworld Korea, PC World Korea; MACEDONIA: PC World Macedonia; MALAYSIA: Computerworld Malaysia, PC World Malaysia, Publish in Asia; MALTA: PC World Malta; MEXICO: Computerworld Mexico, PC World Mexico; MYANMAR: PC World Myanmar; NETHERLANDS: Computer! Totaal, LAN Internetworking Magazine, LAN World Buyers Guide, Macworld Netherlands, Net, WebWereld; NEW ZEALAND: Absolute Beginners Guide and Plain & Simple Series, Computer Buyer, Computer Industry Directory, Computerworld New Zealand, MTB, Network World, PC World New Zealand; NICARAGUA: PC World Centro America; NORWAY: Computerworld Norge, CW Rapport, Datamagasinet, Financial Rapport, Kursguide Norge, Macworld Norge, Multimediaworld Norge, PC World Ekspress Norge, PC World Nettverk, PC World Norge, PC World ProduktGuide Norge; PAKISTAN: Computerworld Pakistan; PANAMA: PC World Panama; PEOPLE'S REPUBLIC OF CHINA: China Computer Users, China Computerworld, China InfoWorld, China Telecom World Weekly, Computer & Communication, Electronic Design China, Electronics Today, Electronics Weekly, Game Software, PC World China, Popular Computer Week, Software Weekly, Software World, Telecom World; PERU: Computerworld Peru, PC World Profesional Peru, PC World SoHo Peru; PHILIPPINES: Click!, Computerworld Philippines, PC World Philippines, Publish in Asia; POLAND: Computerworld Poland, Computerworld Special Report Poland, Cyber, Macworld Poland, Networld Poland, PC World Komputer; PORTUGAL: Cerebro/PC World, Computerworld/Correio Informático, Dealer World Portugal, Mac*In/PC*In Portugal, Multimedia World; PUERTO RICO: PC World Puerto Rico; ROMANIA: Computerworld Romania, PC World Romania, Telecom Romania; RUSSIA: Computerworld Russia, Mir PK, Publish, Seti; SINGAPORE: Computerworld Singapore, PC World Singapore, Publish in Asia; SLOVENIA: Monitor; SOUTH AFRICA: Computing SA, Network World SA, Software World SA; SPAIN: Communicaciones World España, Computerworld España, Dealer World España, Macworld España, PC World España; SRI LANKA: Infolink PC World; SWEDEN: CAP&Design, Computer Sweden, Corporate Computing Sweden, Internetworld Sweden, it.branschen, Macworld Sweden, Maxidata Sweden, MikroDatorn, Natverk & Kommunikation, PC World Sweden, PCaktiv, Windows World Sweden; SWITZERLAND: Computerworld Schweiz, Macworld Schweiz, PCtip; TAIWAN: Computerworld Taiwan, Macworld Taiwan, NEW ViSION/Publish, PC World Taiwan, Windows World Taiwan; THAILAND: Publish in Asia, Thai Computerworld; TURKEY: Computerworld Turkiye, Macworld Turkiye, Network World Turkiye, PC World Turkiye; UKRAINE: Computerworld Kiev, Multimedia World Ukraine, PC World Ukraine; UNITED KINGDOM: Acorn User UK, Amiga Action UK, Amiga Computing UK, Apple Talk UK, Computing, Macworld, Parents and Computers UK, PC Advisor, PC Home, PSX Pro, The WEB; UNITED STATES: Cable in the Classroom, CIO Magazine, Computerworld, DOS World, Federal Computer Week, GamePro Magazine, InfoWorld, I-Way, Macworld, Network World, PC Games, PC World, Publish, Video Event, THE WEB Magazine, and WebMaster; online webzines: JavaWorld, NetscapeWorld, and SunWorld Online; URUGUAY: InfoWorld Uruguay; VENEZUELA: Computerworld Venezuela, PC World Venezuela; and VIETNAM: PC World Vietnam [illegible]

محتويات الكتاب



مقـــدمـــة

إن Delphi 4 Bible يقدم إرشاداً كاملاً ومرجعاً للبرمجة مع Delphi. إنك لا تحتاج الى خبرة سابقة فى مجال البرمجة لكى تستخدم هذا الكتاب، ولكن اذا كنت تعرف القليل عن الـ Pascal أو الـ C أو الـ C++ أو الـ Visual Basic فإن هذا يساعدك الى حد ما. وتقدم تعليمات هذا الكتاب الموضحة خطوة بخطوة تقنيات Delphi من مستوى المبتدئين الى المستويات الأكثر تقدما. إن لغة الـ object Pascal الخاصة بـ Delphi وكذلك الـ Visual Components Library موجودة فى وثائق، وسوف تتعلم كيف تنشئ واجهات تطبيق عملية خاصة بالمستخدم للعديد من التطبيقات. والأمثلة الكثيرة- والمتوفرة مع source code- تنير لك طريقك.

معلومات حول هذا الكتاب:

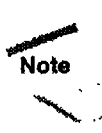
إن الـ Delphi 4 Bible يغطى أوامر الـ Delphi ومميزاته وتقنياته فى أربعة أجزاء:

● **الجزء الأول: مقدمة:** تقدم الأبواب الموجودة فى هذا الجزء البيئة البرمجية الخاصة بـ Delphi وتشرح العديد من أوامره ومميزاته. سوف تتعلم كيفية استخدام الـ visual components والـ forms والخصائص والـ event لإنشاء تطبيقات عملية ذات ٣٢ بت للـ Windows 95 والـ Windows 98 والـ Windows NT.

● **الجزء الثانى: واجهة التطبيق للمستخدم:** توضح أبواب هذا الجزء كيفية إنشاء واجهات تطبيق للمستخدم والـ Visual Components Library الخاصة بـ Delphi. سوف تعرف كيفية برمجة الفأرة ولوحة المفاتيح، وتنشئ واجهة التطبيق للمستخدم باستخدام الـ object مثل القوائم، الأزرار، toolbars، Coolbars، status panels، scroll bar، dialog box، page controls، dockable windows. سوف تتعلم أيضاً كيف تعمل مع strings أحادى السطر، Multiple-Line Text، وتتنقل عبر الأدلة والملفات، وتتصل بـ dialog box.

• **الجزء الثالث: التطبيق:** تغطي أبواب هذا الجزء موضوعات تطوير التطبيق وتشرح كيفية استخدام الـvisual components المتقدمة الخاصة بـ Delphi. سوف تعرف كيفية برمجة تطبيقات للـ Graphics وللصور المتحركة متعددة الوسائط، ومخرجات الطابعة، وتطبيقات MDI، object-linking and embedding وتطوير قاعدة البيانات. سوف تتعلم أيضاً كيف تنشئ تقارير Charts باستخدام الـ visual component الـ QuickReport والـ TeeChart الخاصة بـ Delphi.

• **الجزء الرابع: التقنيات المتقدمة:** تقدم الأبواب الموجودة في هذا الجزء موضوعات شيقة لمطوري Delphi المتقدمين سوف تعرف كيف تستخدم الـ exception لمعالجة الأخطاء، وتستخدم وتنشئ المستندات، وتنشئ custom component مخصصة وتحولها الى ActiveX control. والباب النهائي في هذا الجزء (وفي الكتاب يقدم تقنيات برمجة متقدمة للـ Object Pascal. سوف تتعلم الجديد حول الـ object-oriented مثل dynamic arrays، method overloading، default parameters، file streams وعدة اشياء اخرى متنوعة.



ملحوظة: ان الجزء الثاني والثالث يقدمان معلومات حول معظم الـ Delphi visual component. لكي تجد مرجعاً للـ component بعينها، ابحث في فهرس الموضوعات في آخر الكتاب.

متطلبات:

بالإضافة الى هذا الكتاب، إنك تحتاج الى نسخة من طبعات الـ Delphi الـ Professional أو الـ Client/Server الخاصة بـ Delphi تكون مثبتة في حاسبك الشخصي أو الشبكة الخاصة بك. إنك تحتاج ايضاً الى تركيب Windows 95، أو Windows 98، و Windows NT. ان كل التطبيقات والبرامج التعليمية الموجودة في هذا الكتاب تستخدم الـ Visual environment الخاصة بـ Delphi والعاملة في بيئة الـ Windows. بالرغم من ان كثير من المعلومات الموجودة في هذا الكتاب تنطبق على كل نسخ Delphi، الا انه يجب ان تستخدم Version 3 أو Version 4 للحصول على أفضل النتائج.

Tip

فكرة: تأكد من وجود ملفات الـ source code الخاصة بـ Delphi (غالباً في دليل الـ source الخاص بـ Delphi). إنك تجد إجابات كثيرة لعدد من التساؤلات بتصفح الـ source code لـ component ما. إنني احبذ قراءة ملف source code لمعرفة مزيد من المعلومات حول component أو موضوع ما.

الاختلافات بين نسخ الـ Delphi:

في Delphi4، إنك تفتح الـ Project Options باختيار الـ View|Project Manager ثم تضغط يميناً المشروع (Project 1، مثلاً). هذا يعرض قائمة والتي تستطيع ان تختار منها أمر الـ....Options لفتح الـ Project Options Dialog . استخدم هذه الطريقة اذا كنت تستخدم Delphi4 عندما يخبرك هذا الكتاب ان تختار الـ ......Project Options، ومازال الـ Project Options ايضاً متوفراً من خلال Project Options menu.

في هذا الكتاب افترض انك تعرف كيف تستخدم الـ Windows وانه لديك فأرة أو جهاز شبيه مثل track Ball . وإنني لن اضيع وقتك ووقتي في موضوعات مثل كيفية اختيار اوامر القائمة، ضغط الازرار، والتنقل بين النوافذ. اذا اردت المساعدة في كيفية استخدام الـ Windows، عليك باستشارة واحداً من البرامج التعليمية الممتازة الموجودة في السوق أو ارجع الى المراجع والارشادات التي تأتي مع نظام تشغيل الـ Windows . إنني ضمنت كثير من الافكار حول الاستخدام الفعال لبيئة التطوير المتكاملة لـ Delphi، ولكن لا تنسى ان هذا في الاساس كتاب عن البرمجة. فإن اغلب محتوياته تركز على كيفية كتابة تطبيقات باستخدام Delphi ولغة برمجة الـ Object Pascal .

الاعراف المستخدمة في هذا الكتاب:

ان كل باب يعتبر صغيراً بحيث تستطيع ان تقرأه في جلسة واحدة. واذا انهيت باباً واحداً أو اثنين في اليوم الواحد فإنك سوف تتمكن من اجادة الـ Delphi في حوالي أسبوعين. ابق معي لاسبوع آخر وسوف تصبح خبيراً !

وإنني اعترف، رغم ذلك، ان الساعات المطلوبة غير متوفرة في اليوم لكل شخص ليقرأ ولو كلمة واحدة في مثل هذا الكتاب الضخم. لكي اسعدك على ان

تجد ما تحتاجه بسرعة- وحتى لا تضيع الوقت فى قراءة ما تعرفه بالفعل- فإن كل الابواب تنتهى بثلاث فصول:

- **مشروعات:** لكى تختبر فهمك لمحتويات الباب، جرب نفسك فى المشروعات المقترحة فى نهاية كل باب. يحتوى القرص المدمج الخاص بهذا الكتاب يحتوى على أى ملفات مذكورة فى هذا الفصل.

- **افكار للمستخدم الخبير:** تصفح هذه الافكار الإضافية لزيادة فهمك عن موضوع الباب لتتعرف على تقنيات البرمجة.

- **ملخص:** اقرأ هذا الملخص الموجودة فى نقاط لتنشيط ذاكرتك عن محتويات الباب. يمكنك ايضاً تصفح الملخصات لتجد الابواب التى تغطى الموضوعات التى تريدها.

Tip

فكرة: اذا كنت تعرف الـ Pascal بالفعل او اذا كنت تعرف Delphi، فتصفح اولاً مشروعات، وافكار، وملخص كل باب لكى تساعدك على تحديد ما اذا ما كان يجب عليك قراءة الباب ام لا. من خلال نص الكتاب، فإنك سوف تجد ملحوظات وافكار وتحذيرات اضافية والتى توضح عشرات التقنيات الخاص بـ Delphi. وهذه الفصول- والتى لها icons على الجانب الايسر من الصفحة- يمكن ايضاً ان تساعدك على ان تجد موضوعات محددة عند التصفح.

On the CD-ROM

على القرص المدمج: قرب نهاية كل باب توجد ملخصات عن الموضوعات الموجودة فى الباب. وفى نهاية كل باب، سوف تجد ايضاً مشروعات مقترحة وافكار عديدة للمستخدم الخبير. لمعرفة القصة الكاملة لـ component ما، وابحث عنه فى المحتويات والفهرس. عليك ايضاً بالتعامل مع Online visual componeut المتواجد على القرص المدمج.

وتذكر: ان كل الـ component classes مشتقة من classes أخرى، لذا لكى تفهم كل الخصائص والـ methods المتوفرة لـ component ما، قد يتعين عليك القراءة عن classes عديدة.

أى من نسخ Delphi يمكنك استخدامها؟:

أغلب الأمثلة فى هذا الكتاب، تستخدم Delphi Version 3 ونسخة متقدمة من Delphi Version 4، والتى تعمل فى الـ Windows 95. تنطبق بعض معلومات هذا الكتاب على كل نسخ Delphi ولكن، للحصول على افضل النتائج، يجب ان تقوم بتشغيل Delphi3 أو Delphi4. والعناصر الجديدةأو العناصر التى تعمل بطريقة مختلفة مشار اليها فى النص. اذا كنت تستخدم Delphi Version1 أو Delphi2، فإنك غالباً ستواجه مشاكل فى تشغيل العديد من التطبيقات يمكنك استخدام نسخة حديثة لكى تبدأ. إننى لاارشح استخدام Delphi1 مع هذا الكتاب الا، ربما، للبداية فقط. يمكنك استخدام Delphi2 مع كثير من هذا الكتاب، ولكن احبذ لك التحديث سريعاً الى Delphi3 أو Delphi4 (تعمل اغلب معلومات والتطبيقات بهذا الكتاب مع Delphi3). اما المميزات الفريدة فى Delphi فقد تم الاشادة إليها.

كيف تصل الى الكاتب ؟:

بالطبع، انا لا اقع فى خطأ (انت تصدق هذا، اليس كذلك؟)، ولكن اذا اكتشفت خطأ فى هذا الكتاب، اكتب لى رسالة على عنوان الناشر أو ارسل لى خطاب بريد الكترونى الى TomSwan@copuserve.com. من فضلك إجعل رسالتك قصيرة. من فضلك ابعث بعنوانك.



الجزء الأول
مقدمة

محتويات هذا الجزء:

- الباب الأول: معلومات حول Delphi 4
- الباب الثانى: معلومات حول الـ Visual Components
- الباب الثالث: معلومات حول الـ Forms

كما ستعرف من خلال هذا الكتاب، وبمساعدة برنامج Delphi، يمكنك تطوير تطبيقات يصعب تطبيقها فى وقت قصير بغض النظر عن خبرتك فى مجال البرمجة.

اذا كنت قد بدأت الآن فى تطوير برامج من خلال بيئة الـ Windows، فإنك قد اخترت النقطة الصحيحة لتبدأ منها. أما اذا كنت تعرف معلومات مسبقة عن الـ Delphi أو اذا كانت لديك الخبرة فى كتابة البرامج لـ Delphi فيمكنك قراءة الأبواب الموجودة فى الجزء الأول قراءة سريعة- ولكن لا تهمل قراءتها. فإن تلك الأبواب تشمل محتويات الكتاب وتحتوى على المعلومات التى تحتاج أن تعرفها قبل أن تتناول موضوعات أكثر تقدماً.

اذا كنت تستخدم إحدى النسخ الثلاث السابق لـ Delphi- وخاصة اذا كنت تريد التعامل مع (Delphi)- أنظر جزء "لمحة تاريخية" فى الباب الأول لتقارن بين الخصائص البرمجية للنسخ السابقة لـ Delphi لترى قائمة بالعناصر الجديدة من خلال (Delphi).



الباب الأول

معلومات حول Delphi 4

محتويات الباب:

- لمحة تاريخية
- تطوير تطبيقات الـ Delphi
- البرمجة من خلال الـ Delphi Components
- وظيفة Code insight
- الـ Module Explorer
- الملفات وامتدادات اسم الملف
- الـ Project Manager

إن البداية دائماً تكون أصعب جزء فى عملية تعلم استخدام أداة تطوير برمجيات جديدة. ولكن، على الرغم من ذلك، يمكنك مع Delphi أن تنشئ تطبيقك الأول فى دقائق. فى هذا الباب أقدم لك الخطوط الرئيسية للكتاب، واركز على الخصائص الرئيسية للـ visual programming الخاصة بـ Delphi.

لمحة تاريخية:

لقد مر برنامج Delphi حتى الآن بأربعة أجيال ولقد زادت كل نسخة جديدة من درجة التعقيد. وبسبب التغييرات الكثيرة التى أجريت لـ Delphi، فقد تحتاج بعض التطبيقات القديمة والـ Components السابقة الى مراجعة قبل أن تستطيع أن تتعامل معها من خلال Delphi.

على الرغم من ذلك ، فإن هذه التغييرات تعتبر بسيطة فى غالبية الحالات . فى الواقع ، لقد تمكنت من تطبيق مشروعات هذا الكتاب المتعددة مستخدماً Delphi 4 بلا أية تعديلات رغم أننى كتبتها أساساً باستخدام نسخة Delphi الأولى . وللحق ، إن التطبيقات من خلال هذا الكتاب تعتبر بسيطة نسبياً ، إلا انها تغطى غالبية خصائص البرمجة فى Delphi إلا أنه بالنسبة الى التطبيقات الخاصة بنظم قواعد البيانات أو OLE فإنه يفضل إعادة كتابتها مرة أخرى .

وأحد أسباب التغيير فى كتابة البرنامج والذى يعتبر سهلاً بالنسبة لبيئة Delphi ، هو أن الـ object class قد تم إضافة بعض الخصائص إليها ، ولكن لم يتم تغييرها بصورة رئيسية . ولقد تمت مراجعة الخصائص بالـ object ، وذلك بشكل أساسى فى غالبية الحالات ، ولكن خصائص class لم تتغير بشكل قوى لذلك ، فإن البرامج التى تستخدمها تحتاج فقط الى إعادة عملية compile لإضافة بعض الخصائص الجديدة للتطبيق الخاصة بـ Delphi 4 ، وتعد هذه إحدى المميزات الخاصة بالـ object-oriented programming .

الخصائص الرئيسية فى Delphi 1:

إن النسخة الأولى من Delphi ، والتى تم اصدارها فى نفس الوقت مع Windows 95 ، قدمت ما يسمى visual programming . وبالرغم من وجود الـ(Visual Basic (VB لفترة ، إلا إنها ما زالت تنتج code تحت سيطرة الـ interpreter ولكن الـ Delphi تقوم بعملية الـ Compile للـ Source code .

Note

ملحوظة: ينشئ Delphi 1 تطبيقات للنظم ذات ١٦ Bit ليتم تشغيلها فى برامج Windows 3.1 . وإنشاء تطبيقات للنظم ذات ٣٢ Bit لم تنفذ إلا فى Delphi 2 .

والخصائص الرئيسية لـ Delphi 1 لا تزال موجودة دون تعديل وهذه الخصائص كما يلى :

- **إضافة visual components على الـ form:** تستطيع أن تضع VCL مباشرة على FORM ، وكل component هو عبارة عن object نستطيع أن نحدد معنى الـ component من ناحية البرمجة بأنه Object Pascal class .

• **Object Inspector Window -property editor:** يستخدم لتحديد المواصفات الخاصة بكل الـ components مثل العنوان الخاص بالـcomponent، وحجمها، وكذلك لونها.

• **event editor:** ويتم اختياره أيضاً بواسطة الـObject Inspector- ويتم إضافة الجمل المراد تنفيذها من خلال زر عند تنفيذ مجموعة من الجمل تختار من الـevent editor الـ event property المخصص لتنفيذ حدوث الضغط على الزر.

• **automatic code generation:** Delphi يكون class والتعريف للمتغيرات وevent-handler procedure فارغة عند بدء تكوين تطبيق جديد، أضف objects على الـ form وأختر الـ object properties لتكوين الـ event handlers والتى تملأ بالـ code الخاص بك .

• **Native .exe code files:** التطبيقات عند إجراء عملية compile تحتوى على كافة VCL التى تنتمى الى Project، مما يزيد حجم الملف ذا الإمتداد exe وأيضاً، يجعلها فى بعض الأحيان مضيعة لمساحة القرص، ولكنها سريعة للغاية، وأصغر بكثير من التطبيقات الشبيهة التى تعمل بـ C++ class libraries والتى تستطيع إنتاج ملفات ضخمة ذا الإمتداد exe، (إن Delphi تحتوى على smart linker ولغة برمجة جيدة يؤدى الى إنتاج تطبيقات جيدة) وعلى النقيض من الـ VB، أنه لا توجد حاجة الى إنتاج run-time interpreter عند استخدام أى تطبيقات كتبت بـ Delphi.

• **إختفاء الـpointer:** لم تختفى الـ Pointers تماماً فى الواقع، فهى ما زالت تستخدم وبشكل قوى فى الـ Object Pascal. ولكنه لم يعد ضرورياً استخدام pointers بشكل كبير فى معظم الحالات. السبب الرئيسى لهذا هو أنه مع وجود بينة VCL، وأصبح التحكم فى الذاكرة مسئولية الـ component libraries بدلاً من واضع البرامج. لذا، فنادراً أن يكون من الضرورى تخصيص ذاكرة Objects واستخدام pointers للتعامل معها. فإن Objects يتم إنشاءها عند الحاجة، وهذا يضع الـ pointers فى عالم النسيان ولكن ما زالت موجودة عند الحاجة الى استخدامها.

● **إمكانية استخدام الـ Visual Basic (VBX):** لا يمكنك إنشاء الـ VBX من خلال Delphi 1، ولكن يمكنك استخدام VBX، وذلك من خلال تحويلها الى component تستطيع التعامل معه.

● **إنشاء نظم قواعد البيانات باستخدام visual components والـ Borland Database:** إن إمكانية كتابة تطبيقات متميزة خاصة لإنشاء كافة عناصر نظم قواعد البيانات تعد العنصر الرئيسى فى نجاح Delphi الساحق.

هناك نسختان متوفرتان من Delphi 1–، هما الـ desktop والـ client-server. ونسخة الـ desktop مصممة لتطوير تطبيق standalone Windows التى لا تتعامل مع الشبكات. ولكن نسخة client-server، تعطيك إمكانية التعامل مع الشبكات، هذا بالإضافة الى الوصول لـ mainframe database من خلال تطبيقات Delphi. ولقد أصبح Delphi الآن متوفر بثلاث نسخ الـ desktop، professional، والـ client-server.

إن الإنتقال من البرمجة باستخدام الـ Turbo Pascal و Borland Pascal الى Delphi 1 يتطلب إعادة النظر فى عملية الإنتقال. حيث، أن غالبية التطبيقات يجب إعادة كتابتها من البداية، مع إحتمال استخدام أجزاء من التطبيقات من خلال الـ Delphi، هذا بالإضافة الى استخدام Delphi لمفهوم visual environment واستخدامه لـ VCL. فإن Visual Component Library (VCL) التابعة لـ Delphi 1 ليست متوافقة مع الـ Borland Pascal ObjectWindows والـ TurboVision السابقة لها. ولهذا السبب، فإن استخدام تطبيق كتب بهما من خلال Delphi 1 هو أمر غاية فى الصعوبة.

علاقة OWL's بـ Delphi

من الناحية الفنية، كان Delphi 1.0 متوافقة بنسبة ١٠٠٪ مع OWL و Turbo Pascal for Windows (TPW). فى الواقع كان OWL تصدر مع Delphi 1.0 حتى يتمكن المطورين من إجراء عملية compile لبرامجهم بلا تعديلات. ولكن لتحويل تطبيقات OWL الى تطبيق يعمل مع Delphi التى تستخدم الـ VCL الجديدة كان هذا يعنى إعادة كتابة غالبية التطبيقات من البداية، الأمر الذى دفع معظم المطورون الى عمله.

هناك سببان لإنفصال Borland عن طراز الـOWL. السبب الأول هو أنه لم يكن ملائماً تماماً لمفهوم (VCL) ولأنه يفتقد إمكانية التعامل مع properties وmethods- وهي مكونات هامة للـ Components. أضف الى ذلك أن OWL كان يحتاج الى تطوير كبير لينتقل الى نظام الـ ٣٢ بت، لذا اختار Borland الخيار المنطقي، وإن كان مؤلماً، في أن ينفصل عن الـOWL وأن يقدم (VCL) مع خصائص الـ Object Pascal لتدعيمها. (في بعض أبواب هذا الكتاب، وعلى القرص المدمج، سوف تجد أيقونة الـOWL الأصلية الخاصة بي والتي كانت معي منذ الـ TPW 1.0).

وفي ملاحظة أخيرة، إن تطبيقات DOS التي كتبت مع الـ Turbo Pascal قد تم تحديثها بسهولة الى تطبيقات الـWin32 باستخدام Delphi 4.

اذا كنت تستخدم Delphi 1، فسوف تجد بعض المعلومات في هذا الكتاب مفيدة لك. ولكن، عليك إجراء تغييرات لكثير من البرامج الموجودة، وبالطبع، الخصائص الموجودة في النسخ التالية غير متاحة لك. ولا اقترح عليك استخدام Delphi 1 مع هذا الكتاب إلا لمجرد البدء فقط.

الخصائص الرئيسية في Delphi 2:

إن الميزة الرئيسية في Delphi Version 2 هي تدعيم إمكانية البرمجة بنظام ٣٢ بت. والخصائص الجديدة الأخرى في Delphi 2 تتضمن:

- **Long and short strings:** أخيراً تم التخلص من مشكلة الـ Borland Pascal- وهي الـ string ذو ٢٥٥ حرف طولاً والذي يطلق عليها "length-byte" وهذا النوع للـ string ما زال موجوداً، ولكن long strings حررت واضعي البرامج أخيراً من الإضطرار الى عدد لا حصر له من الأساليب الكثيرة لتخزين long strings في الذاكرة.

- **32-bit integer:** والذي جعل لا بد من إعادة النظر الى المتغيرات التي يكون الـ ١٦ بت طولاً لها.

- **All cod and data references are now far (32 bits):** وهذا لا يؤثر على برمجة التطبيقات إلا في الحالات التي تكون فيها مستخدماً للغة assembly- وهذه تكون خطيرة على البرنامج.

- **الأماكن التى يتم حجزها فى الذاكرة محدودة بكم الذاكرة المتاح فقط والذى قد يكون ذا حجم كبير من الـ megabytes.**
- **Huge arrays:** يصل حجم تكوينها ما يوازى حجم 32 bit index.
- **استخدام ما يسمى بالـ multithreaded.**
- **التعامل مع مكونات الـ Windows ذى الـ ٣٢ بت مثل TreeView و outlines.**
- **إلغاء إمكانية التعامل مع الـ VBX ذات الـ ١٦ بت،** وإضافة إمكانية التعامل الـ OCX ذات الـ ٣٢ بت.
- **تحسين إمكانية التعامل مع الـ Object Linking and Embedding (OLE).**
- **تحسين إمكانية التعامل بين Borland Database Engine و SQL.**
- **صدرت عدة نسخ مثل الـ desktop، developer والـ client-server.**
- **التعامل فقط مع Windows 95 أو NT. حيث لم تعد التطبيقات الخاصة بهذه النسخة تعمل مع Windows 3.1.**

من خلال مسار التطور، من Delphi 1 الى Delphi 2 لن تجد صعوبة لأن الـ classes الموجودة لم تتغير الى حد كبير. وإن السبيل الى تحديث تطبيق ما هو إلا أن تحول تفكيرك الى نظام الـ ٣٢ بت ويتم تحديث أى برنامج من حيث pointers و 16-bit integer، ويجب تحديث أية procedures أو functions إلى الـ ٣٢ بت. على الرغم من ذلك، هذه الخصائص الجديدة تعتبر أكثر آلية لغالبية واضعى البرامج الذين يستخدمون visual environment لـ Delphi لإنشاء التطبيقات.

فى غالبية الحالات، يعتبر إعادة عملية الـ compiling للبرنامج وكذلك components هو كل ما يحتاج إليه للإرتقاء من Delphi 1 الى Delphi 2.

ولا ينطبق هذا على تطبيقات Delphi 1 التى تستخدم الـ VBX ذات الـ ١٦ بت، والذى قامت Borland بتقديمها كعينات غير متوفرة فى Version 2. وهذا



يتضمن البرامج التى تستخدم مكونات الـ TBiSwitch ، TBiGauge ، TBiPict ، والـ TChart .

Caution

تحذير: إن تجربتنا مع الـ VBX ذات الـ ١٦ بت تشير الى أنه ليس من الحكمة استخدام Delphi Sample components فى تطوير التطبيق . إننى هنا أشير الى الـ component الموجودة VCL palette Samples وكل الأمثلة من البرامج والتى تأتى مع Delphi ، ننوى استخدامها فى البرنامج النهائى .

اذا كان تطبيقك يستخدم أية resources ذات النظام ١٦ بت ، فسوف تحتاج أن تعيد عملية الـ compile لها وباستخدام linker (32-bit) لتكون الملف ذى الإمتداد (exe) النهائى . على سبيل المثال ، اذا قمت بإدخال cursor bitmaps ، فعليك استخدام الـ Image Editor لإنشاء resources ذى ٣٢ بت جديد وذلك لإحتواء التطبيق للـ cursor images .

يمكنك استخدام Delphi 1 مع هذا الكتاب ، ولكن اقترح عليك الإرتقاء بأسرع ما يمكن الى Version 3 أو Version 4 .

الخصائص الرئيسية فى Delphi 3:

تستطيع أن تلقب هذه النسخة بأنها النسخة التى عززت مركز Delphi كأعلى نظام تطوير فى الـ Windows . والخصائص الجديدة فى Delphi 3 تتضمن :

- **code packages:** والذى مكن التطبيقات المتعددة من التشارك فى كتابة البرنامج . والفائدة الرئيسية هى أنه لم يعد ضرورياً لكل تطبيق تمت له عملية الـ compile أن يحتوى على الـ VCL ، كما كان الحال فى النسختين السابقتين . الـcode packages تعنى أن مطورى الـ components يجب أن يراجعوا منتجاتهم ويعدوهم للاستخدام من قبل المستخدم- وأصبحت عملية تطوير بناء component كتطوير التطبيقات تماماً .

- **Numerous enhancements to the VCL:** وتشمل Win32 components الجديدة . وكذلك تتوفر الـ charting و third-party .

- **تدعـيـم كـامل لـ ActiveX، بما فى ذلك إنشاءها؛** ويضـيف Delphi 3 إمكانية وجود لـ native language مما يسهل استخدام وإنشاء الـ COM Objects من خلال Delphi.

- **القـدرة على أداء عـمليـة compile للـ units وتحـويلها الى الملفـات ذات الإمتـداد (obj.):** هو تغيير طال انتظاره منذ عهد الـ Turbo Pascal. وهذا يسـمح للـ compiled module بأن يتم إدخـالهـا فى نظم تطوير أخـرى مـثل الـ C++.

- **component client-server و database multitiered تم تدعـيـمـهـا من خـلال TClientDataset والـ TRemoteServer الجديدة؛** وهذه الـ component تتوفر فى طبعة الـ client-server فى Delphi3 فقط.

- **عنـاصـر الـ Internet و intranet تم تكوينهـا من خـلال نسـخـة الـ professional والـ client-server؛** وتتـصل هذه component بالـ server الخـاص بشبـكة Web وتقـوم بتبـسـيط تطوير تطبـيق الـ Web وذلك بأن تخلص واضعى البرامج من ضرورة الإلمام HTML communinicaton protocols.

- **بنية تطوير متكاملة ويطلق عليها (IDE) والـ code insight؛** وهذا يضيف خصائص كانت فى الماضى متاحة لمحررى البرمجة الثانويين فقط. ولكن شكل Delphi على الرغم من ذلك لم يتغير كثيراً.

- **تدعيم إمكانية WideString character-string type؛**

- **الـربـط بـين الـ Borland Database Engine و SQL التـى تم مراجعتها حديثاً للحصول على تطوير قاعدة بيانات متطورة؛**

- **إصـدارات الـ Delphi مـثل الـ Standard أو الـ professional والـ client-server؛** ان غالبية الـ components وادوات تطوير قاعدة البيانات تتوفر فى جميع الإصدارات. ولكن طبعة الـ client-server توفر ادوات لإنشاء تطبيقات database multities. بإسـتـخـدام الـ remote SQL servers والشـبـكات

المحلية . وتوفر نسخة الـ professional امكانات إنشاء تطبيقات على الشبكات المحلية وكذلك توفر الدعم للتطبيقات التى تستخدم ODBC بواسطة third-party drivers .

Note

ملحوظة: فى الواقع ، ان عالم Delphi 3 له بعد رابع . اذا كان لديك شغف للمعرفة فيمكنك الحصول على طبعة الـ Delphi Enterprise ، والتى تتضمن مزيد من الادوات لتوصيل تطبيقات Delphi 3 بمصادر قاعدة البيانات الى "data sources" باستخدام تكنولوجيا الـ enterprise الخاصة بـ Borland . أما السعر ؟ فعليك ان تسأل . . . . . .

لتحديث تطبيقات Delphi 2 الى Version 3 ، فيمكنك كالعادة إعادة عملية الـ compile دون صنع أية تغييرات . ولكن ، قد تريد الإنتفاع بمزايا الخصائص الجديدة مثل إمكانية الـ Internet وربما ، الـ ActiveX .

وبعض الـ Delphi 3 components موضوعة فى أقسام مختلفة تحتويها VCL palette . وبعض الـ component s الأخرى قد تم استبدالها بـ components مشابهة ، وإن كانت غير مطابقة . على سبيل المثال ، إن Header component لم يعد متاحاً على Additional component ، بل تم استبداله بـ HeaderControl ذا ٣٢ بت ولكن يفتقد الى بعض خصائص الـ component السابق .

يمكنك استخدام Delphi 3 مع غالبية المعلومات والأمثلة الموجودة فى هذا الكتاب . والخصائص الموجودة فى Delphi 4 فقط دون غيره مميزة بوضوح أو مشار إليها فى النص .

الخصائص الرئيسية Delphi 4:

إن مستخدمى أى من نسخ Delphi السابقة يجب أن يكونوا قادرين على الإرتقاء مباشرة الى Delphi 4 . والخصائص الجديدة فى Delphi 4 تشمل البنود التالية ، ومعظمها موضح تفصيلياً من خلال هذا الكتاب :

● **شكل جديد، ولكنه مألوف:** إن Delphi speed Bar و VCL palette ، واللذان كانا فيما مضى واقعين الى جوار بعضهما البعض ، أصبحا الآن فوق

بعضهما. ولقد أصبح هذا ضرورياً لأن عدد component والأقسام قد زاد عن حجم الـ palette الأساسي بشكل كبير. مع Delphi 4، لم يعد من الضروري تحريك اللوحة يميناً ويساراً لتجد الـ component التي تحتاجها. (قد يظل هذا ضرورياً مع low resolution).

• **امتدادات لغة Object Pascal**: وهذا يؤدي الى عدة تغييرات في الـ Object Pascal، ولكن جميع التغييرات ذات طبيعة متطورة ومن غير المحتمل أن تقوم بإيقاف تنفيذ البرنامج الموجودة. وبعض التغييرات تجعل الـ Object Pascal أكثر شيوعاً بالـ C++ وأكثر قيمة لمطوري component الذين يريدون استخدام نسخ Delphi و C++ Builder لمنتجاتهم. وخصائص اللغة الجديدة تتضمن dynamic arrays، method overloading، وقيم ثابتة، والنوع الجديد 46-bit Integer، في النوع Real، والتغيير في بعض الأساليب الفرعية لإستدعاء Windows API وإستدعاء الثوابت أيضاً. انظر الباب الحادي والعشرين، تطوير مهارات Delphi الخاصة بك، للحصول على معلومات حول خصائص الـ Object Pascal.

• **Project Manager**: ان مدير المشروع القديم قد تم إعادة إنشاءه كاملاً في صورة tree-outline model، مما يمكنك من متابعة مسار ملفات المشروع بسهولة. ولقد اصبح من الممكن الآن ايضاً إنشاء مجموعات مشروع، والتي بامكانها ان تقوم بتصنيف مشروعات متعددة بطريقة متطورة ويتم تخزين مجموعة المشروع في صورة text Make file ويمكن بواسطة هذا ملف إجراء عملية compile على المشروعات من نظام التشغيل الـ "DOS" ولمزيد من المعلومات انظر فقرة "Project Manager" في هذا الباب.

• **Module Explorer**: ان code editor التابع لـ Delphi يتضاعف في الـ IQ مع هذه الإصدارة الجديدة. والـ Module Explorer، الذي يوجد عادة الى اليسار من نافذة code editor ولا يمكن نقلها الى نوافذة منفصلة، ببساطة يتواجد في ملف pascal source code unit ويظهر الـ Module Explorer كافة التقارير في unit file، التي يمكنك استقدامها لتحديد مكان إجزاء متنوعة من البرمجة في النص. على سبيل المثال، ان الـ Module Explorer يمنحك طريقة

بسيطة للانتقال من تعريف method الى تنفيذ ذلك الـ method. لمزيد من المعلومات انظر فقرة "Module Explorer" فى هذا الباب.

• **Dockable Tool Windows**: ان نوافذ ادوات Delphi تعتبر Dockable. على سبيل المثال، يمكنك Undock toolBar أو نافذة اخرى، مثل الـ Module Explorer، وتعيد عرضها كنافذة مستقلة عن الحاسب. يمكنك ايضاً ان تصل نوافذ لإنشاء نوافذ متعددة الصفحات. على سبيل المثال، بدلاً من عرض نوافذ الـ Breakpoints والـ Watches منفصلة، يمكنك توصيلها فى نافذة واحدة ثم اختر الادوات الفردية بالضغط على ابواب الصفحات ببساطة قم بضغط وسحب النافذة التى تريد توصيلها أو فصلها. ملحوظة: لتوصيل نافذة، اضغطها واسحبها الى حافة نافذة أخرى، مثل الـ code editor واستمر فى السحب حتى يتغير حجم الخطوط الخارجية للنافذة، وفى أى نقطة يمكنك ترك الفأرة.

• **Improved Debugging**: هناك ثلاثة عناصر إضيفت لعملية debugging.

نافذة CPU تظهر الـ low-level state لتطبيق ما، بما فى ذلك قيم الـ register و flag الـ CPU. يمكنك ايضاً مراجعة وإزالة اخطاء تطبيقاتك على مستوى لغة اذا كانت assembly. وتظهر نافذة Module الآن الـ module الخاصة بالتطبيق، قائمة بملفات التى تمت عملية الـ compile لها لإنشاء module وقائمة glonal symbols و entry points وتقدم النافذة الثالثة الجديدة نافذة logEvent التى توضح رسائل process control ورسائل breakpoints، رسائل الـ Output-Debug-String، ورسائل النافذة. ويحتوى Delphi 4 ايضاً خصائص إزالة اخطاء متقدمة مثل remote debugging، multiprocess debugging، cross-language debugging، debugging of CORBA.

• **زيادة عدد VCL**: لقد إزدادت عدد VCL components فى Delphi 4. وهى تشمل قوائم أوامر لربط الاستجابات الخاصة بالـ objects من الناحية الوظيفية مثل القوائم والازرار، امكانية وجود بعد التطبيقات خاصة بالـ Windows NT، وعناصر الـ TControl والـ TWinControl لإنشاء نوافذ و toolbars والتحكم عند حدوث تغيير فى حجم النوافذ أو عند حدوث تغييرات أخرى وتتضمن نسخة

الـ client-server الـ component TQueryClientDataSet الجـــــديدة وcomponent أخرى .

• **برمجة Multitiered Database**: تزيد الـ components الجـديدة من قدرة برمجة بيانات الـ client-server. فقد تم تحسين عملية الاتصال من خلال Socket server mconnect unit والتى تدعم الـ call backs وsocket server application يتم تنفيذها كـ Windows NT . server . وتشمل خصائص قاعدة البيانات المتقدمة الأخرى امكانية وجود الـ Oracle8 Abstract Data Types (ADTs) كحقول للـ Table.

• **دعم Microsoft Transaction server (MTS) component**: ان database classes فى Delphi 4، و COM class، وحتى بيئة التطوير المتكامل لـ Delphi تساعد على انشاء، تركيب، وdelugging للـ com objects وتحت سيطرة MTS. وهناك أيضاً امكانية انشاء، إزالة اخطاء والتعامل مع CORBA objects.

• **Database Programming Enhancements**: يمكنك تصفح قواعد البيانات وتعديل الـ Tables بشكل ايسر باستخدام dataset TTable component . إضغط زر الفأرة اليمين لـ TTable object لتتمكن من الوصول الى field editor ولكى تحذف، وتعيد تسمية، وتقوم بتحديث هياكل الـ Table. يمكنك أيضاً تشغيل الـ Database Explorer من هذه القائمة . اذا كان الـ Table متصلاً بمصدر بيانات، فهذا يعرض الجدول ومعلوماته فى الـ Explorer.

• **ما تم إضافته الى ActiveX**: لقد قامت Delphi بزيادة كفاءة الـ ActiveX لتشمل لـ Visual Basic data interfaces وأيضاً توفير إمكانية إنشاء nonvisual ActiveX.

اذا كان لديك Delphi4، فيمكنك استخدام جميع المعلومات وتشغيل كل الأمثلة الموجودة فى هذا الكتاب . يمكنك استخدام أى من إصدارات Delphi (أو professional) أو (desktop)، فأنت لا تحتاج الى الـ client-server، رغم أننى قد ذكرت بعض قدراتها . (لقد دارت شائعة أن Inprise سوف يغير أنواع

الإصـــدارات المطروحـــة مع Delphi4 ولكن هذا لن يؤثر على المعلومــات التى ستحصل عليها من هذا الكتاب) .

Note **ملحـوظة:** إن المعلومـات الموجـودة فى هذا الكتـاب حـول الخصـائص الجديدة فى Delphi 4 يمكن أن تساعدك فى تحديد اذا ما كنت سوف تقوم بالتحـديث من نسخـة سابقـة ، هذا اذا لم تكن قد اتخـذت هذه الخطوة بالفعل أو تخشى المخاطرة بها . اذا لم يكن لديك Delphi 4 حتى الآن ، فيمكنك استخدام فى Delphi 3 مع غالبية الأمثلة الموجودة فى هذا الكتاب .

تطبيقات Delphi المطورة:

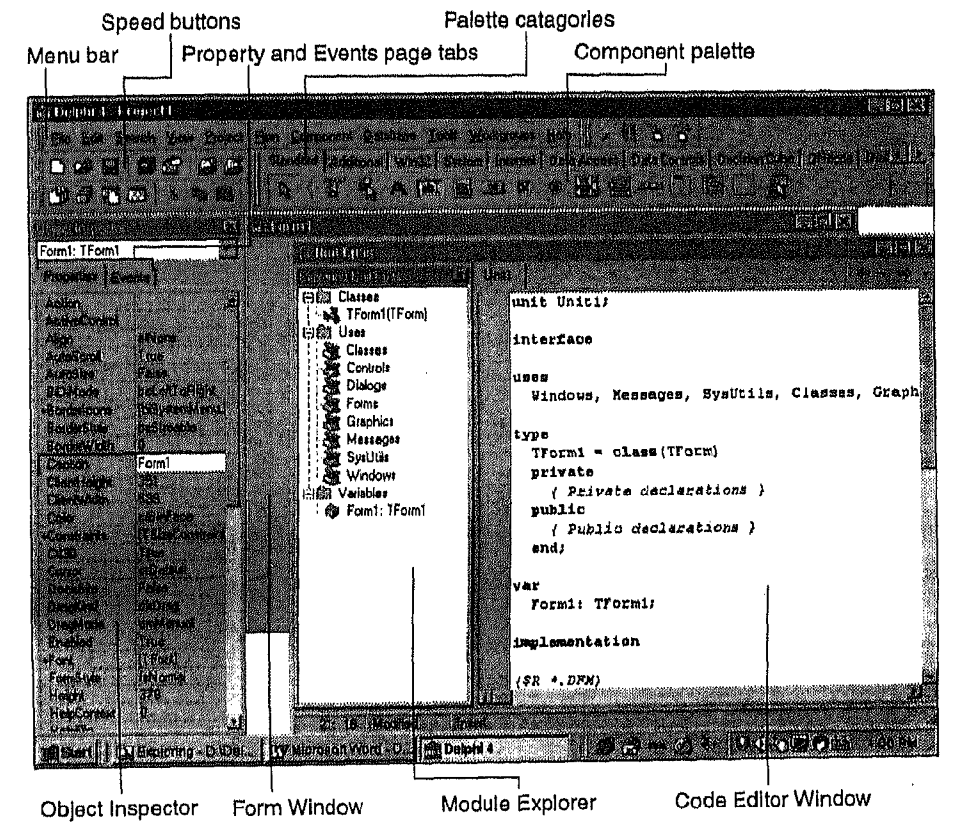
كـمـا هو الحـال مع مـعظم برامج الحـاسب ، إن أفـضل طريقـة للتعـرف على Delphi هو اسـتـخـدامـه . فـفى هذه الفـقـرة ، سـوف تقـوم بتـجـربة visual environment الخـاصــة بـ Delphi . إبدأ الآن مع Delphi يوضح شكل (١-١) Delphi وهو يعمل فى Windows 95 اذا لم تنشئ شاشتك الشكل ، اختر أمر الـ File | New Application .

قد تختلف بعض الأزرار والنوافذ وهذا يعتمد على نسخة برامجك ، فنظام التشـغيل ، وعناصـر عـمليـة installation . واسـتناداً الى نسـخـة الـ Windows الخاصة بك ، قد تـختلف كثير من title bars ، borders ، buttons أشياء أخرى عن تلك الموضحة هنا . وتلك الفروق إنما هى فروق شكلية الى حد كبير ، ولا تؤثر على أى من المعلومات الواردة فى هذا الكتاب إلا فى المواضع التى أشرت إليها .

Delphi [integrated development environment (IDE)] لهـا عـدة عناصر أساسية . وتلك العناصر هى [حسب الترتيب الوارد فى شكل (١-١)] :

- **Speed buttons:** هناك مـا يسـمى point-and-click buttons وهى مخصصة لإختيار الأوامر من القوائم . يمكنك أن تعرف ماذا يفعل الزر وذلك بوضع مـؤشـرة الفـأرة عليـه (ولكن لا تضـغط) وتنتظر لحظة لتـرى الـ hint box والذى يحتوى على اسم الزر . تمرن على استخدام الـ Speed Button الخاصة بـ Delphi والتى على المدى البعيد سوف توفر وقتك الذى تبذله فى عملية التطوير ، وخاصة اذا كنت تتحكم فى الفـأرة بشكل جيـد . على سبيل المثال، بالرغم من أنه بإمكانك

تشغيل برامج بطرق أخرى، إن أسرع وسيلة هي في الغالب أت تضغط زر الـ "Run" (المثلث المشير الى اليمين). تم تغيير بعض الأيقونات في Delphi 4 لتتلائم بشكل أفضل مع توصيات الـ Microsoft على سبيل المثال، زر المشروع Save (الثاني من اليسار) يبدو الآن وكأنه مجموعة من الأقراص.



شكل (١-١): Delphi 4 وهو يعمل في الـ Windows 95

- **Menu bar:** وهي قائمة تتشابه في الشكل مع القوائم في بيئة الـ Windows. قم بفحص قوائم Delphi الآن لكي تتعرف عليها. إن Delphi 4 يضيف أيقونات لكثير من الأوامر. إنها نفس الأيقونات المعروضة في شكل Speed buttons في الـ toolbar.

- **Component palette:** تحتوى هذه palette على أيقونات تمثل الـ component في الـ VCL. اضغط palette button لتختار visual



component ثم اضغط الفأرة داخل الـ form window لتقوم بإضافة component object للحصول على معلومات عن component معين، ضع مؤشر الفأرة على أى component button ولكن لا تضغط الفأرة. بعد لحظة، يقوم Delphi بعرض hint box يحتوى على اسم الـ component قم بتحريك الفأرة لترى الـ hint boxes عن أى component أخر. استخدم هذا الأسلوب لتحديد مكان component معينة متكررة فى هذا الكتاب.

- **Palette categories:** لرؤية المزيد من palette categories، اختر عنصراً من عناصر page tabs والذى يظهر بصورة واضحة أعلى VCL palette. على سبيل المثال، اختر عنصر Dialogs page لكى يظهر لك Delphi's dialog-box component.

- **Properties and Events page tabs:** اختر واحدا منهما وهما إلى الأعلى من Object Inspector window إما لتختر properties window أو event window، الـ properties window تمثل الخواص الخاصة للـ component، مثل حجم الزر أو تحديد نوع الخط للـ text label. و event window تمثل حدوث شئ ما، مثل الضغط لإحدى أزرار الفأرة أو ضغط أى زر من المفاتيح.

- **Object Inspector:** توضح هذه النافذة كل الـproperties والـevents لواحد أو أكثر من الـ component أو forms المختارة. بمرور الوقت سوف تصبح صديقاً حميماً لهذه النافذة، فبالرغم من بساطتها الظاهرية، تعد نافذة الـ Object Inspector واحدة من أهم أدوات البرمجة الخاصة بـ Delphi. (فكرة: اضغط F11 لإستدعاء نافذة الـ Object Inspector اذا كانت مختبئة بنافذة أخرى، أو اذا كانت أغلقت فجأة).

- **Form:** فى كل التطبيقات تمثل الـ form الأولى الـ form الرئيسية للبرنامج ومايلى من نوافذ، تمثل نوافذ أخرى- على سبيل المثال dialog box أو a child window فى Multiple Document Interface (MDI). يمكن ان يكون هناك form وحيدة للبرامج البسيطة أما التطبيقات المعقدة فيكون لها عشرات الـ forms وتساعدك الـ dotted grid على تنظيم المسافات بين components

المضافه الى الـ form . ولا تظهر الـ dotted grid فى التطبيق النهائى . (فكرة: أضغط F12 مرة واحدة أو أكثر لتحديد موضع هذه النافذة اذا كانت تخفيها نافذة أخرى).

• **الـModule Explorer:** وهو الجديد فى Delphi 4 ، فهذه النافذة تظهر الـ current module الحالية بالإضافة الى قائمة بالـ units الأخرى ، التى من الممكن التعامل معها من خلال إستدعائك لـ variables أو objects أو methods، وأى معلومات أخرى. للتوصل الى الـ method فى الـ code editor window، فعليك ببساطة ان تضغط أى من العناصر المدخله للـ Module Explorer.

• **Code editorwindow:** تظهر هذه النافذة البرنامج المكتوب بلغة Pascal المرتبطة بكل form فى التطبيق. وسوف اطلق عليها code editor أو اسم editor window من الآن فصاعداً ويقوم Delphi بصورة تلقائية بإضافة التعريف والـ events الى هذه النافذة، والتى يمكنك ان تضيف إليها جمل Pascal أو التى تنفذ مباشرة تبعاً للـ events الذى يحدث مثل أوامر القائمة أو الـ button clicks. يمكنك أيضاً استخدام هذه النافذة لتحرير Pascal modules أو عرض text files قم بالاختيار من بين الملفات المفتوحة بإختيار أحد عناصر الـ page tabs المتواجدة أعلى الـ editor window .

• **Windows 95 task bar :** يقوم الـ Windows 95 task bar بعرض زرStart وايقونات التطبيقات التى يتم تحميلها حالياً. ولإعادة التعامل مع تلك البرامج المحملة، عليك ان تضغط الزر الخاص به بتلك البرامج.

Tip **فكرة:** اضغط F 12 لتربط بين code editor، form windows ، التى تختبئ وراء بعضها البعض.



سوف تعرف اكثر عن النوافذ، الأوامر والامكانات الاخرى فى Delphi كلما قمت بتجربة الأمثلة الموجودة فى هذا الكتاب. كما انك ستقابل الكثير من النوافذ الأخرى مثل الـ Project Manager ، الـ Object Browser ، الـ Integrated Debugger، الـ Code Editor ، الـ Code Insights ، الـ

Image Editor، الـ Package Editor ، الـ Menu Designer ، وادوات اخرى . حتى تصبح اكثر تعرفاً على Delphi ، يجب ان تعيد ترتيب عرض شاشتك حتى تشبه الشكل (١-١) .

Tip

فكرة: قبل ان نستمر ، اختر Tools|Environment Options واختر Preferences page tab ، وأختر عنصر component captions هذا يعطى معلومات أكثر عن الـ components رغم ان واضعى البرامج ذوى الخبره يرفضون هذا الخيار . ان Delphi لايظهر captions على جميع الـ components ولكن فقط على أولئك الذين ليس لهم شكل متميز . على سبيل المثال ، مع اختيار هذا الخيار ، يتم تسمية DataSource component (الذى تستخدم فى برمجة قاعدة البيانات) بـ DataSource1 أو باسم آخر اذا قمت بتغييره . وبدون caption على component فسوف تظهر component على انها ايقونات ، مما يجعل من الصعب التمييز بينهم .

إعداد تطبيق جديدة:

ان الخطوة الأولى بعد البدء فى تطبيق جديد هى ان تعطيه اسماً . اننى احب ان اقوم بهذا عقب اختيار الـ File|New Application مباشرة . ان تسمية مشروع جديد باسرع وقت ممكن يمنع Delphi من حفظ المشروع فى أدلته تحت اسماء ملفات افتراضية للبرنامج ، مما يهدر مساحة القرص .

ويتكون مشروع Delphi المثالى من عدة انواع من الملفات . تحتوى بعض الملفات على text ، واخرى تحتوى على binary values ، bitmaps ، codes . ولان كل تطبيق يتكون من عدة ملفات ، فإنه من الصواب إنشاء دليل منفصل على قرصك الصلب لكل مشروع جديد . بهذه الطريقة . يمكنك بسهولة نسخ ملفات تطبيق على قرص مرن أو محرك شبكة لحفظه بطريقة آمنة ، ويمكنك ان تحذف التطبيقات القديمة بإزالة الأدلة التابعة لها .

Tip

فكرة: ان حفظ المشروعات فى أدلة منفصلة يساعد على حفظ application modules بشكل منظم . لا تقم بتخزين مشروعات متعددة فى نفس الدليل - ان المجموعة الناتجة من الملفات سوف تصبح من الصعب فصلها عن بعضها .

لإنشاء اول تطبيقاتك فى Delphi، قم باداء الخطوات التالية الآن :

١ - استخدام الـ Windows Explorer لإنشاء دليل جديد (يمكنك ان تسميه مجلد) مثل C:\Projects لتخزين مشروعات Delphi الخاصة بك. يمكنك استخدام drive غير الـ :C واسم آخر اذا كنت تفضل هذا. وفى داخل دليلك الجديد، قم بإنشاء دليل فرعى اسمه Hello. والمسار التام، C:\Projects\Hello جاهز الآن لحمل ملفات المشروع.

٢- إرجع الى Delphi واختر FileNew Application . وبالتبادل، يمكنك اختيار FileNew، اضغط Page tab ، واختر ايقونة الـ Application - ولكن هذه هى الطريقة الطويلة للوصول لنفس النهاية. اذا تم تحميل وتعديل مشروع آخر، فإن Delphi يعطيك فرصة لحفظ المشروع او للتخلص منه.

٣- قبل إضافة component أو عمل أى تغييرات أخرى فى المشروع الجديد، اختر FileSave All (اضغط Alt+FV)، أو اضغط Save All button. إنك ترى two dialogs، واحداً تلو الآخر. الأول، والذى يحمل اسم Save Unit1 As، يطلب اسم ملف لملف source code الرئيسى الخاص بالـ Unit. واحب ان اطلق على هذا الملف اسم Main، وهذا يشير الى انه يحتوى على source code الخاص الـ form الرئيسية للقيام بهذا، افتح الدليل الذى انشأته فى الخطوة الأولى (Step 1)، واكتب Main داخل مربع ادخال اسم الـ File الخاص Save dialog. اضغط Enter أو اضغط Save لإنشاء ملف الـ Main.pas فى الدليل المفتوح. ويقوم Delphi تلقائياً بإلحاق امتداد اسم الملف pas. اذا لم توفر واحداً. ولا تقم بتغيير هذا الامتداد أو أى امتداد آخر بلا تمييز. ان Delphi يتعرف على انواع الملفات من امتداداتها، واذا قمت بتغييرها، فان Delphi قد لا يكون قادراً على انشاء التطبيق النهائى.

٤- ثم يظهر Delphi بعد ذلك dialog آخر يحمل اسم Save Project1 As. وأود ان اطلق على مشروعاتى نفس اسماء أدلتها. على سبيل المثال، فى هذه الحالة، ادخل Hello لإسم ملف المشروع، ثم اضغط Enter أو اضغط Save. وهذا يخلق ملف المشروع الرئيسى، Hello.dpr، فى الدليل المفتوح يوفر Delphi تلقائياً امتداد اسم الملف dpr، والذى ترمز الى .Delphi Project .

Caution

تحذير: اذا لم يطلب Delphi اسم المشروع فى الخطوة الرابعة، فقد يكون بطريق الخطأ قد اخترت أحد أمرى الـ File|Save الأخرى، والذى قد لا يقوم بحفظ المشروع بأكمله. اذا حدث ذلك، فببساطة كرر الخطوات ٣ و ٤ . فى الخطوة ٣، يمكنك اختيار Save Project As...(Alt+FE) File| ، لكننى أرى انه من الايسر حفظ مشروعاتى وأى ملفات متعلقة باختيار .File|Save All (Alt+FV).

لتطوير مزيداً من تطبيق Delphi الأول الخاص بك، استمر مع التعليمات الموجودة فى الفصل التالى.

Tip

فكرة: لضمان ان كل ملفات المشروع قد تم حفظها، اختر الـ Tools|Environment Options، اختر Preferences page tab ، قم بتشغيل خيارى الـ Autosave وهما ملفات Editorm والـ Desktop. بهذه الطريقة، فإن أية تغييرات للملفات المشروع يتم حفظها تلقائياً فى كل مرة قبل ان تقوم بتشغيلها. لا تختر هذه الخيارات رغم ذلك، اذا أردت عمل تغييرات مؤقتة وقمت بإدارتها لإختيار نظريات وافكار.

تحديد الـ window caption:

سوف تجد ان Delphi قد وضع مسمى للـ caption وهو Form1 وهو خاص للـForm الرئيسية للمشروع Hello وتجد دائماً الـ caption عبارة عن جزء من أى Form ويظهر دائماً فى الجزء العلوى للـ Form.

ولتغيير هذا المسماة نتبع الخطوات التالية:

١ - يجب ان تعرض نافذة الـObject Inspector Form1: TForm1 فى أعلاها- اذا لم يكن الأمر كذلك، اضغط داخل نافذة Form1 (تلك التى تحتوى على dotted grid). اضغط F11 و F12 كما تحتاج لإستدعاء النوافذ اللازمة لتراها. فى أعلى نافذة الـ Object Inspector [ارجع الى الشكل (١-١)]، اضغط Properties page tabs لعرض الـ properties form، وهذا إيضاً يختار نافذة الـ Object Inspector فتستطيع صنع تغييرات للـ events وللـ object . properties

٢- اختر خاصية الـ Caption . (ممكن ان يكون قد تم اختيارها بالفعل حسب النظام الافتراضى للبرامج) .

٣- لتعديل خاصية ما ، اكتب أو اختر قيمة جديدة فى العمود الواقع الى اليمين من اسم الخاصية . على سبيل المثال ، قم بتغيير قيمة الـ Caption من Form1 الى Hello Delphi Programmer . فقط إبدأ الكتابة - وعليك ان تضغط Enter بعد الكتابة لاحظ ان caption يتغير وانت تكتب .

٤- احفظ المشروع باختيار File|Save All . ولأنك قد قمت بالفعل بإعطاء اسماء ملفات ، فان Delphi ينفذ الأمر على الفور - أى انه لا يعرض save dialog التى رأيتها عندما قمت بإنشاء المشروع أول مرة . ولكن ، فى كل مرة تضيف فيها Form جديدة للمشروع ، فسوف ترى واحداً أو أكثر من هذه الـ dialog مرة أخرى عندما تحفظ المشروع .

Tip

فكرة: عندما يتم حفظ المشروع تماماً ، فإن Save All speed button سوف تصبح فى حالة dimmed . وهذه طريقة سريعة للتأكد بصرياً من ان أى تغييرات للـ project modules قد تم تخزينها بشكل سليم على القرص .

لإنهاء تطبيقك الأول ، استكمل التعليمات فى الفصل التالى .

تشغيل التطبيق:

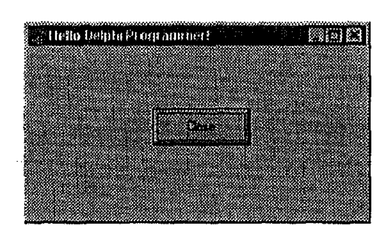
صدق أولاً تصدق ، إنك قد انهيت من برمجة تطبيق Delphi الأول الخاص بك لتكملة عملية التطوير وتشغيل البرنامج ، اتبع هذه الخطوات :

١- اضغط F9 ، أو اختر امر الـ Run من قائمة الـ Run أو ، إضغط Run button (المثلث المواجه اليمين) . استخدم احدى هاتين الوسيلتين لتشغيل تطبيقك مراراً خلال التطوير ، ولاختيار البرمجة الجديدة . ليس عليك ان تنهى مشروعاً قبل ان تشغله .

٢- ان تروس حاسبك تعمل ، يدور القرص ، يخرج دخان من تجويفات محرك القرص ، وعقب ثوان قليلة ، تظهر نافذة التطبيق الاول الخاص بك على الشاشة . يوضح شكل (١-٢) عرض Hello فى الـ Windows 95 . قد يختلف

مظهر البرنامج إعتماداً على مواصفات نظام التشغيل الذى حددته عند إنشاءك لنظام التشغيل .

٣- لإنهاء التطبيق ، اضغط مرتين زر قائمة النظام فى الركن الأيسر العلوى فى النافذة ، أو اضغط مرة واحدة الزر واختر Close من قائمة النظام . اذا كنت تفضل استخدام لوحة المفاتيح ، اضغط Alt+F4 لإغلاق التطبيق وعد الى Delphi . أو ، يمكنك ضغط زر إغلاق النافذة فى الركن الأيمن العلوى .



شكل (١-٢): عرضت تطبيق Hello

رغم أن تطبيق Hello لا يفيد كثيراً ، إلا أنه يعرض جانب مهم من برمجة Delphi . إن المشروعات تبدأ العمل كتطبيقات تامة . قارن هذا بنظام الـ nonvisual ، مثل تركيب الـ C++ التقليدى الذى قد يتطلب منك أن تكتب عشرات ، أو أكثر ، من السطور قبل أن تتمكن من البدء فى إختيار برنامجك . إن برامج Delphi مستعدة للتشغيل فى غالبية المراحل الخاصة بتطويرها .

Caution

تحذير: بعد تشغيل التطبيقات من داخل Delphi ، أغلقهم دائماً لترجع الى نمط البرمجة . اذا تركت التطبيق مفتوحاً بالمصادفة ، فلن تكون قادراً على إختيار كثيراً من أوامر Delphi . اذا لم تستطيع التعامل مع Delphi – وخاصة اذا لم تستطيع زؤية نافذة الـ Object Inspector بضغط الـ F11 – فإن السبب من المحتمل ان يكون هو انك نسيت وتركت التطبيق الذى يعمل بدون إغلاق .

ابدأ تجربة أخرى سريعة : اضغط F9 لتعود الى Hello . لاحظ انه هذه المرة ، تظهر نافذة التطبيق اسرع من أول مرة قمت فيها بتشغيل البرنامج . يدرك Delphi أنك لم تقم باية تغييرات فى المشروع ، لذا يقوم ببساطة بإعادة تحميل ملف الـ application code مباشرة . اغلق نافذة برنامج Hello الآن .

يمكنك ايضاً استخدام الـ Windows Explorer لتشغيل تطبيق Delphi . على سبيل المثال، افتح الـ Explorer، وتصفح دليل الـ C:\Projects\Hello . اضغط مرتين Hello.exe، أو قم بابراز اسم الملف واضغط Enter . ان مستخدمى برنامجك يستطعون إنهاء التعامل مع الـ Delphi كما يفعلون مع برامج الـ Windows الأخرى . تذكر اغلاق البرنامج قبل الاستمرار .

من الممكن ان تستخدم الـ Windows Explorer لتشغيل أكثر من نتيجة للبرنامج على سبيل المثال، يمكنك تشغيل نسختين أو أكثر من برنامج Helleo . ويمكنك، رغم ذلك، ان تقوم بتشغيل البرنامج مرة واحدة فقط من داخل Delphi . لتشغيل البرنامج عدة مرات، يجب ان تستخدم امر الـ button Run . أو الـ Explorer . اغلق كل نسخ البرنامج Hello قبل الاستمرار .

ان تطبيقات التى تم إنتاجها بواسطة Delphi تتمتع بصفات بيئة الـ Windows كاملة - فانك تقوم بتشغيله بنفس الطريقة التى تديرها التطبيقات الأخرى . يمكنك ان تضغط وتسحب اسم ملف خاص بالتطبيق desktop، أو ان تنشئ . shortcut للملف (اضغط واسحب مستخدماً زر الفأرة الأيمن)، ثم اضغط مرتين الأيقونة الناتجة . وأولئك الذين يحبون ان يؤدون الأشياء بالطريقة الصعبة يمكنهم . كتابة اسم المسار الكامل للبرنامج C:\Projects\Hello\Hello.exe باستخدام أمر الـ Run فى زر Start . يمكنك أيضاً ان تفتح نافذة DOS، وان تنتقل الى دليل المشروع، وان تدخل Hello لتشغيل البرنامج . الفكرة هى ان التطبيقات التى انشأها Delphi تعتبر برامج له نفس التشابه فى التنفيذ كما يحدث فى بيئة الـ Windows حيث أنك تقوم بتشغيلها بنفس الطريقة التى تشغل بها برامج الـ Windows الاخرى .

عملتى الـ (Compiling) والـ (linking) الخاصتان للبرنامج:

عندما تقوم بتشغيل تطبيق بضغط الـ F9، يقوم Delphi بعملية ثم عملية الـ compiles والـ link للبرنامج لإنشاء executable code file . ويحدث شيئان اساسيان الأول هو ان compiler الخاص بـ Delphi يترجم نص البرامج الى binary code . ثم يقوم الـ linker بضم binary code الى الـ modules الأخرى المطلوبة لأعمال البدء المتنوعة ومهام أخرى . ونتيجة ما حدث لمشروع Delphi ينتج

ملف من نوع executable ويحمل نفس اسم المشروع ، ولكن ذو اسم ملف ينتهى بالإمتداد exe .

لإجراء عمليتى الـ compiling و الـ linking لتطبيق ما ، ولكن دون تشغيله ، اضغط Ctrl+F9 أو اختر أمر الـ Project|Syntax Check . يمكنك استخدام هذه الطرق للتأكد من أن البرنامج خالى من الأخطاء مثل أخطاء الكتابة البسيطة .

Tip

فكرة: استخدم الـ Tools|Environment وقم بإختيار Show compiler progress . وهذا يعرض dialog يوضح التفاصيل المتنوعة لعمليتى الـ compiling والـ linking وهو مفيد خاصة لتحديد الأخطاء فى التطبيقات الكبيرة .

وعلى عكس النظم الأخرى التى تتمتع بحزم الـ visual programming ينتج Delphi (true native code) ، وهذا يعنى أن البرامج التامة لا تتطلب مترجم وقت التشغيل ففى النسخ السابقة من Delphi ، كان الملف exe. الناتج تام بنسبة ١٠٠٪ ، وكان هو الملف الوحيد المطلوب من قبل مستخدمى برنامجك . وهذا ما زال ينطبق على Delphi 4 . ولكن ، مع بداية Delphi Version 3 ، تم ترتيب كثير من Delphi's Visual Component Library (VCL) فى package يمكن لتطبيقين أو أكثر التشارك فيها . وإعتماداً على أنواع التطبيقات التى تقوم بإنشاءها ، فقد تحتاج الى run-time packages وقت التصميم أو وقت التشغيل (أو كلاهما) للمستخدمين النهائيين . إنك قطعاً سوف تحتاج الى فعل هذا ، على سبيل المثال ، اذا قمت بتطوير custom components لواضعى برامج Delphi آخرين .

وسبب آخر لتوزيع package هو لتقليل كمية الـ code فى ملفات الـ exe. ففى نسخ Delphi Versions 1 and 2 ، كانت كل ما يحتويه المشروع من code ، بما فى ذلك الـ code الخاص بـ VCL بأكملها ، قد تم تجميعها فى كل ملف exe. . اذا كان لديك ستة تطبيقات Delphi مختلفة ، فإن ملفات الـ exe. الخاصة بهم تكون خزنت ست نسخ منفصلة من نفس مكونات الـ VCL . ومن الواضح أن إضاعة المساحة هذه أمر غير مرغوب فيه ، رغم أن الملفات الناتجة تكون بسيطة وسريعة التشغيل . واليوم ، يمكنك إختيار أن توزع الـ code العامة مستخدماً

package تتشارك فيها جميع التطبيقات المضيفة . ولا يوجد أية معوقات لسرعة استخدام package (كما هو الحال Visual Basic و Java اذ يستخدم run-time interpreters . ويوضح الباب االعشرون ، إنشاء Components مخصصة عن كيفية إنشاء الـ package الخاصة بك .

Note

ملحوظة: إن Packages تعد أشياء إختيارية على مستوى التطبيق فى وقت التشغيل . اذا كنت تريد تجميع كل الـ codes المستخدمة من قبل تطبيقك ، اختر ...Project|Options ، اضغط Packages page tabs ، ولا تختر الإختيار المسمى Build with runtime packages .

برمجة الـComponent:

قبل الإستمرار فى هذا الباب ، دعنا نتوقف قليلاً لنتذكر سريعاً هذه الأشياء . إنك تعرف الآن ثلاث تقنيات هامة لـ Delphi :

* كيفية إنشاء وحفظ مشروع جديد .
* كيفية تعديل خاصية مثل الـ Caption .
* كيفية عمل compile ، وعمل link ، وتشغيل تطبيق بضغط F9 ، أو بضغط الزر Run (المثلث المشير الى اليمين) .

إنك سوف تستخدم هذه التقنيات فى كل تطبيق تقوم بكتابته . وبالطبع ، إن البرامج التى لا تفعل شيئاً سوى عرض عنوان لن تأخذ شعار الأكثر مبيعاً . ولإعطاء تطبيق Hello شيئاً ليفعله ، يمكنك إدخال visual component ، فى نافذة البرنامج ، وقم بإختبار قمة الـ code ، كما ستوضح الفقرات التالية .

Tip

فكرة: يقوم Open-file speed Button بفتح الملفات الفردية . ولتحميل تطبيق Delphi كامل ، اضغط دائماً speed Button Open-project ، أو اختر الـ ...File|Open لفتح ملف مشروع ينتهى بالإمتداد dpr.. يمكنك أيضاً إختيار أمر الـ ...File|Open Project ، والذى ظهر ثانية فى Delphi 4 بعد إختفاءه من Version 3 .

إضافة visual components:

اتبع هذه الخطوات لإضافة visual components فى نافذة برنامج Hello :

١- أغلق برنامج Hello العامل اذا كان ضرورياً للرجوع الى Delphi. اختر الـ...File|Open أو ضغط زر Open-project . (قم بهذا لإكتساب خبرة حتى اذا كان Hello مفتوحاً بالفعل). عندما يظهر Open Project dialog، قم بتغيير الأدلة اذا لزم الأمر واختر Hello.dpr. اضغط Enter أو اضغط OK لفتح المشروع استخدم نفس الطريقة لفتح المشاريع الأخرى المذكورة فى هذا الكتاب.

Tip **فكرة:** اذا كنت تحتاج الى التوقف قليلاً أثناء قراءة الباب، احفظ المشروع ثم قم بإعادة تحميله عندما تستعد للإستكمال.

٢- اختر Standard page tab من components palette ثم اختر Button component الذى يحمل علامة Ok.

٣- حرك الفأرة داخل الـ form واضغط مرة لإضافةالـ Button object ولا يهم الموقع.

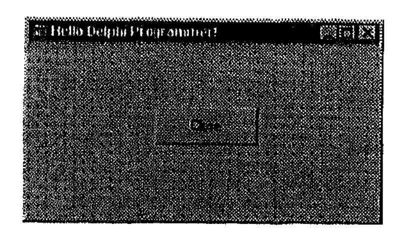
٤- تأكد من أنه تم إختيار الـButton object- يجب أن ترى أشكال مربعة تحيط به. اذا لم يتم إختيار الـobject، اضغطه مرة الآن. وبعد ذلك يقوم Delphi بتسمية الـobject بـ Button1. وللحصول على اسم يحمل معنى أكثر من هذا، اختر خاصية الـ Name فى الـObject Inspector، واكتب CloseButton. إنها كلمة واحدة وليست إثنتان- لا تضع أية مسافات بينها. اضغط Enter.

٥- لاحظ أن خاصية الـ Caption ونص الزر يتغيران أيضاً ليصبحا CloseButton. وهذا أمر طبيعى، ولكنك سوف تحتاج دائماً الى التفرقة بين الـ component object's name وcaption properties، على سبيل المثال، اختر خاصية الـ Caption واكتب Close. ويجب أن تشبه الـ form الخاصة بك شكل (١-٣).

٦- اضغط مرة واحدة CloseButton object اذا لزم الأمر لتختاره. وفى أعلى نافذة الـ Object Inspector، اختر Events page tab، والتى تضع قائمة بالأعمال التى يمكن للزر القيام بها. اضغط مرتين المساحة الخالية الى اليمين من

حـدث الـ OnClick . يظهـر الـ Main.pas فى الـ code editor ، ويضع Delphi المؤشر الوامض بين الكلمتين الرئيسيتين بداية begin ونهاية end . اكتب العبارة الآتية :

```
Close;
```



شكل (١-٣): Hello's form فى Delphi

٧- لقد قمت الآن ببرمجة procedure يدعى OnClick event handler الخاص بالزر . تأكد من إنهاء العبارة بفصلة منقوطة . قارن النص الموجود فى نافذتك بذلك الموجود فى القائمة (١-١) .

القائمة (١-١): برنامج OnClick event handler لـ CloseButton object

```
procedure TForm1.CloseButtonClick(Sender: TObject);
begin
  Close;
end;
```

٨- احفظ المشروع ثم قم بتشغيله بضغط الـF9 . يمكنك الآن ضغط زر الـ OK لإغلاق النافذة . ولأن هذه تعد نافذة البرنامج الرئيسية ، فإن إغلقها ينهى التطبيق ايضاً .

ان الخطوات السابقة توضح ثلاث جوانب هامة للبرمجة مع Delphi :

* إنك قمت بإضافة Button component object على الـ form .

* إنك قمت بتعديل خصائص object لتغيير قيم الـ Name والـ Caption الخاصة به .

* إنك قمت ببرمجة واحداً من الـ object event لإداء فعل فى وقت التشغيل استجابة لإختيار الزر .

ان معظم خبرتك للبرمجة مع Delphi تتضمن استخدام component objects, properties، و events . لإنشاء التطبيقات، قم بإضافة object component على الـ form ، وقم ببرمجة خصائصهم وبرمجة الـ event handler الخاصة بهم. انه أمر غاية فى السهولة. وبالمناسبة، ان واجهة التطبيق لـ Delphi وجميع components تمت كتابتها باستخدام Delphi- وهذا اثبات قوى لما تستطيع تنفيذه مع نظام التطوير المتميز هذا.

إزالة الأخطاء بجمل البرنامج:

ان واضعى البرامج لا يحبون عملية إزالة الاخطاء. من منا لا يفضل تصميم code جديد عن إصدار الوقت فى إزالة الاخطاء؟ ان Delphi لا يستطيع منع الاخطاء بالطبع، ولكنه يستطيع معاونتك على تجنب كثير من الاخطاء الشائعة وذلك بإعطائك visual environment تقوم تلقائياً بإنشاء جزء كبير من تطبيقك. للأسف، مازال هناك فرصة كبيرة لحدوث الخطاء، ولن اكون صادقاً اذا قلت ان البرمجة مع Delphi تكون خالية من الاخطاء.

اوامر إزالة الاخطاء الخاصة بـ Delphi تنفيذ البرنامج سطر بسطر وراجع قيم الـobjects والمتغيرات. وبإبطاء السرعة المعتادة للبرنامج، يظهر بوضوح سبب المشكلة. تمرن على اوامر ازالة الاخطاء الخاصة بـ Delphi الآن حتى تعرف كيف تستخدمهم حينما تحتاج إليهم - صدقنى، إنك سوف تحتاج إليهم.

وتعتبر أوامر إزالة الاخطاء الخاصة بـ Delphi مفيدة ايضاً لاستكشاف كيف يعمل البرنامج. عليك استخدامها باستمرار لاختيار الأمثلة الموجودة فى هذا الكتاب وكذلك تلك الموجودة فى دليل Demos التابع لـ Delphi. اتبع الخطوات التالية لتعلم كيفية استخدام بعض خصائص إزالة الاخطاء التابعة لـ Delphi:

١- افتح مشروع Hello اذا لزم الأمر.

٢- اذا لم تكن قد أضفت زر Close على الـ form افعل ذلك الآن باتباع التعليمات الموجودة فى الفقرة السابقة.

٣- حدد موضع الـ Form1.CloseButtonClick procedure فى الـ window editor الخاص بـ Main.pas. قم بادخال العبارة التالية فوق Close

لعرض رسالة والسطران الواقعان بين البداية (begin) والنهاية (end) يجب ان يكونا كالاتى:

```
ShowMessage('Ready to close application');
Close;
```

٤- من الطبيعى ان تقوم بضغط F9 لإجراء عمليتى الـ compile، والـ link، تشغيل البرنامج. ولكن هذه المره اضغط F8 لاختيار امر الـ Step Over الخاص بـ Delphi من قائمة الـ Run. هنا يبدأ البرنامج، ويتوقف عند العبارة الأولى (فى الحقيقة هذا يعتمد على نسخة Delphi التى تستخدمها فمن الممكن ان يتوقف عند unit's initialization، اضغط F8 مرة اخرى اذا حدث هذا) يجب ان ترى نافذة البرنامج. فعندما تظهر، اضغط زر الـ Close. وبدلاً من إنهاء البرنامج، فإن Delphi يقوم بايقافه داخل الـ Close Button Click procedure وهذا يعد مثال للـ single-stepping (أو الإنتقال من سطر إلى سطر)، وهو واحداً من اهم ادوات إزالة الاخطاء المستخدمة اثناء تطوير برنامج.

٥- اضغط F8 مرتين اخرتين لتنقل الى عبارة الـ ShowMessage وتنفيذها. يظهر مربع له زر OK على الشاشة. اضغط هذا الزر. ومرة أخرى، لإنك ضغطت F8 فإن Delphi يقوم بايقاف البرنامج مؤقتاً فى العبارة الواقعة بعد التنفيذ وهى Close.

٦- اضغط F9 لتنفيذ عبارة الـ Close وقم بتشغيل البرنامج حتى النهاية. بعد ضغط الـ F8 لتخطى عبارة أو أكثر، فإنك سوف تستمر غالباً فى تشغيل البرنامج بهذه الطريقة بضغط الـ F9. ينتهى البرنامج، ويعود Delphi من نمط ازالة الأخطاء الى نمط التطوير.

وطريقة اخرى للتخطى داخل برنامج هى بتحديد الـ breakpoint. عندما يصل البرنامج الى العبارة المشار إليها، فإن Delphi يقوم بإيقاف التطبيق قبل ان يتم تنفيذ العبارة. اتبع هذه الخطوات لتحديد الـ breakpoint ولتتخطى داخل البرنامج.

١- قم بتحريك المؤشر المضئ الى عبارة الـ ShowMessage فى الـ Main.pas، واختر Run|Add Breakpoint. اضغط زر الـ New فى

button ، واضغط Enter . والعبارة المشار إليها يتم الآن ابرازها باللون الاحمر (أو اللون الرمادى الداكن على الشاشات احادية اللون). أو، يمكنك توجيه مؤشر الفأرة الى اليسار من العبارة واضغط زر الفأرة لتحديد الـ breakpoint استخدم هذه الطريقة لإنشاء أو إلغاء الـ breakpoints . يمكنك تحديد اكثر من breakpoints واحدة فى اماكن مختلفة فى البرنامج ، ولكن فى هذه التجربة ، فإننا ستقوم بتحديد واحد فقط .

٢- اضغط F9 لتشغيل البرنامج . عندما تضغط زر الـ Close يقوم Delphi بإيقاف تنفيذ الـ code عند العبارة المشار إليها . اضغط F8 لتخطى هذه العبارة أغلق message dialog بضغط زر OK الخاص به واضغط الـ F9 لتشغيل البرنامج حتى اتمامه .

٣- جرب الخطوة رقم ٢ مرة أخرى ، اضغط Alt+F4 لإنهاء البرنامج . ولان هذا لا يؤدى الى تنفيذ الـ CloseButtonClick procedure ، فإن البرنامج ينتهى دون التوقف عند الـ breakpoint . ملحوظة : النسخ الاولى من Delphi تعرض ايقونة علامة توقف عندما تقوم بتحريك المؤشر فوق نافذة برنامج متوقف . أما النسخ الأحدث لم تعد تفعل ذلك .

٤- حاول دائماً تشغيل برنامج حتى إنهاء تنفيذه كاملاً واذا لم تتمكن من فعل هذا ، اختر Run|Program Reset (Ctrl+F2) وابدأ العمل . وهذا يؤدى الى عدم جعل Delphi ينتهى عمليات غلق المشروع بصورة صحيحة وإلغاء ما قد يكون فى الذاكرة ، مما يؤدى الى فقد تدريجى للذاكرة المتاحة التى تتطلب منك فى النهاية ان تخرج وتعيد البدء فى Windows . تلك هى حياة مطور البرامج ، ولكن لحسن الحظ هذه المشكلة نادرة الحدوث .

٥- لإنشاء أو إلغاء الـ breakpoint ، حرك مؤشر الفأرة الى اقصى اليسار من العبارة المشار إليها واضغط الزر الايسر مرة . تلك هى اسهل طريقة لإنشاء أو إلغاء الـ breakpoint يمكنك ايضاً استخدام امر View|Breakpoints لإضافة ، أو لحذف ، أوإبطال أوأداء عمليات اخرى على الـ breakpoint . اختر هذا الامر الآن واضغط زر الفأرة الايمن فى نافذة قائمة الـ Breakpoint للحصول على قائمة من الاوامر المتاحة . قم بإزالة الـ breakpoint من عبارة الـ ShowMessage الآن وذلك باختيار امر الـ Delphi من القائمة .

وسأقدم لك اوامر اخرى لإزالة الاخطاء خاصة بـ Delphi وذلك فى الاوقات المناسبة خلال هذا الكتاب . ولكن لا تنتظرنى لاقترح عليك تخطى عبارات برنامج وذلك بضغط الـ F8 وتحديد الـ breakpoint . عليك استخدام خصائص إزالة الاخطاء فى Delphi باستمرار وذلك لتصحيح الأمثلة فى هذا الكتاب . وانك سوف تنبهر بما ستعرفه فقط بإبطاء سرعة تنفيذ الـ Code واختيار حالة البرنامج فى مواضع استراتيجية .

:(Code Insight)

من الاشياء الجديدة والتى تم تقديمها فى Delphi 3 هو ان الـ code insights تقوم تلقائياً بتكملة code اثناء قيامك بعملية الكتابة على الحاسب ، كما انها توفر المعلومات حول component libraries و functions و procedures التابعين لـ Delphi باستخدام الـ code insights ، يمكنك إنشاء مجموعة من عناصر البرمجة العامة مثل الـprocedure والـ loop يمكنك حينئذ ملئ المجموعة لحفظ وقت الكتابة . يمكنك ايضاً الحصول على الـ Delphi's code editor الذى يعرض انواع الـ parameters لهذه procedures و functions ويوضح ايضاً خصائص و methods و component objects ان توفر هذه المعلومات فى code editor يساعدك على اختصار الوقت المبذول فى البحث من خلال online help الخاصة بـ Delphi وهذا اذا ما عرفت كيفية استخدام code insights جيداً . وهذه الفقرة سوف تساعدك .

فى بعض الاحيان ، اجد ان الـ code insights تشكل مشكلة اكثر من فوائدها- وخاصة على الحاسبات الاقل سرعة والتى تتوقف لفترة طويلة قبل عرض on line help الخاصة بالمضمون . على الناحية الأخرى ، إننى اجد الـ code insights غير ذى قيمة كلما استخدمها . على سبيل المثال ، لإنشاء procedure جديد ، إننى اقوم بكتابة كلمة procedure واضغط Ctrl+J ، وهذا يؤدى الى إنشاء procedure كامل والذى يمكننى من ملأه . وباستخدام انواع اخرى من code insight - code completion ، code parameter . وفى الفقرات التالية ، أوضح لك أمثلة لكل تلك التقنيات .

من المحتمل انك تريد ان تجعل الـ code insights ملائمة لإسلوب برمجتك ، ولكن قبل ان تقضى وقتا طويلاً فى صنع هذا ، من الافضل ان تتأقلم

على المواضيع الموجودة حسب نظام البرنامج وخصائص code insights الاربعة الاساسية هي :

* Code Templates
* Code Completion
* Code Parameters
* Tool-tip Expression Evaluation

والفقرات التالية تعطينا ارشادات لاستخدام code insights وكذلك مقترحات لمعرفة كيفية العمل بادوات البرمجة القيمة هذه .

Code templates:

اضغط الـ Ctrl+J في أي وقت وانت تستخدم code editor لعرض قائمة لـ code templates. إختر template الذى تريد ، ويقوم Delphi تلقائياً بإضافة الـ template فى source code فى الاتجاه الحالى للمؤشر ولان هناك مؤشرات عديدة ، يمكنك اختصار الوقت بكتابة الجزء الأول من templates الذى تريد قبل إضغط الـ Ctrl+J . على سبيل المثال ، لإدخال عبارة if فى الـ code ، اكتب كلمة if واضغط Ctrl+J . يضع Delphi أربع من الـ (code templates) الممكنة الآتية فى قائمة :

if statement	ifb
if then (no begin/end) else (no begin/end)	ife
if then else	ifeb
if (no begin/end)	ifs

يصف العمود الأول نوع الـ statement template . ويعطى العمود الأخير اسم الـtemplate مختصراً ، والذى يمكنك استخدامه لعرض template اذا لزم الأمر . اذا لم تكن تريد عرض template يمكنك تجاهل اسماء المختصرة .

إنك تعرف المزيد عن الـcode template باستخدامها أكثر من مجرد القراءة عنها هنا لذا لن احاول ان اذكر الأنواع الكثيرة المتاحة اقض بعض الدقائق فى اختيار الـ template مختلفة للعبارات التى تنوى استخدامها فى الغالب . اننى اجد ان

بعض الـ template اكثر فائدة من غيرها . على سبيل المثال، إننى استغرق وقتاً اطول لتحريك المؤشر حول if-then-else أكثر من لو أننى قمت بكتابة الكلمات الأساسية مباشرة - ولكننى اكتف بسرعة ولكننى أجد template أخرى اكثر فائدة، خاصة لأننى أيضاً استخدم الـ C++ ولغات برمجة أخرى . على سبيل المثال، ان code templates تذكرنى ان الـ exceprion Object Pascal تستخدم الكلمتين الاساستين try و except ، بينما لغات أخرى قد تستخدم try و catch . ان الـCode insights تساعدنى على التنبيه إلى تجميع الـ Object Pascal Syntax، وتخفض من اخطاء أثناء عملية الـ compilation .

فكرة: لا تضع مسافة بعد الجزء الأول من الـ code templat . اكتب فقط الكلمة الأولى من العبارة - على سبيل المثال if ، أو for أو while- ثم اضغط Ctrl+J. يضيف Delphi مسافات حسب تصميم الـ template .

بعد أن تتعود على استخدام الـ code template، قد تريد أن تقوم بعرض المجموعة المخزنة وقد تريد أيضاً أن تنشئ الـ template الخاصة بك . ولأغراض العرض، سوف تقوم بإنشاء procedure template جديد والذى يسبقه تعليق يمكنك أن تملأه لوصف الـ code template ولفعل ذلك اتبع هذه الخطوات :

١- اختر Tools|Environment Options واضغط Code Insight page tab. اضغط زر الـ Add، وأدخل procedureC كالاسم المختصر لـ template، و "procedure (commented)" كوصفه .

٢- اضغط داخل editor window الواقعة بعد علامة الـ Code . أدخل السطور التالية أو استخدام تصميمك الخاص (يشير vertical bar الى المكان الذى تريد أن يضع فيه Delphi مؤشر النص بعد إدخال template فى code البرنامج) .

```
{ Purpose: |
  Date   : 00/00/00
     Author : Tom Duck
}
procedure ();
```

```
begin

end;
```

يمكنك الآن إضافة template آخر أو يمكنك عرض الـ template الموجودة . اضغط OK لحفظ تغييراتك . لاستخدام الـ template ، اكتب P ثم اضغط Ctrl+J . هذا يقدم لك قائمة باسماء جميع الـ template التى تبدأ بحرف الـ P . اختر template الجديد المسمى procedureC لإدخال الـtemplate الى الـcode editor .

ملحوظة: اذا كان هناك template واحد عندما تقوم بضغط الـ Ctrl+J ، يقوم Delphi مباشرة بإدخال template الى code editor . باتباع الخطوات السابقة ، عليك الآن الاختيار من بين إثنين من الـ template . لكلمة الـ procedure الرئيسية . وللعودة للوضع الإفتراضى ، افتح Code Insight dialog ، اختر template الجديدة (procedureC) ، واضغط زر الـ Delete . يمكنك الآن أن تكتب الـ P وتتبعها بـ Ctrl+J لإدخال الـ template procedure مباشرة الى Code . وبالمثل ، يمكنك كتابة F يتبعها Ctrl+J لإضافة function template .

:Code completion

إن إدارة code completion الخاصة بـ Delphi أيضاً تقوم بإدخال النص تلقائياً إلى code editor ، ولكنها تختلف عن code templates بأنها تعطى الـmethods والخصائص والـ events والـ arguments . إن هذه الخاصية توفر الوقت وخاصة فى أنها تساعد على منع أخطاء الـ compiler أكثر من مجرد توفير وقت الكتابة . على سبيل المثال ، يمكنك وضع كل الـ methods الممكنة لأى VCL class object فى قائمة . وبالإختيار من هذه القائمة ، فأنت أقل عرضة للوقوع فى خطأ فى التراكيب ، أو تعيين قيمة خاطئة أو محاولة استدعاء methods غير موجودة .

وإننى أجد code completion مفيد بشكل خاص فى برمجه الـ objects من خلال الـ source code فبدلاً من البحث عن اسماء الخصائص والـ methods

عن طريق online help الخاصة بـ Delphi، فإنني اكتب جزء من الـcode وانتظر لحظة حتى يظهر الـ code completion insight عندئذ أقوم بإختيار الطريقة أو أى بند آخر أريده وأستمر فى الكتابة قدر ما يوفر وقت البحث.

اتبع الخطوات التالية لتجربة code completion ولترى كيف تختلف عن الـcode templates:

١- إبدأ تطبيق جديد بإختيار File|New Application. يمكنك تسمية وحفظ التطبيق اذا أردت ذلك، أو يمكنك استخدام الاسماء الموجودة.

٢- أضف الـ Button component من Standard component palette الى الـ form الخاصة بالتطبيق. اضغط مرتين الـ button object لتكوين event handler.

Tip

فكرة: هذه هى أسرع طريقة لإنشاء event handler لـ object ما، بالرغم من أنه بإمكانك استخدام نافذة الـ Object Inspector كما هو موضح بالنسبة لتطبيق Hello فى هذا الباب.

٣- يجب وضع المؤشر بين كلمة البداية الرئيسية وكلمة النهاية الرئيسية فى الـButton1Click procedure استخدم الآن code completion لتغيير button's label عند ضغطه. اولاً، اكتب الآتى:

```
Button1.
```

ثم انتظر لحظة. لا تضع مسافة بعد النقطة وسوف ترى اقتراحات code completion لهذا النوع من الـ object. (اذا اختفت المقترحات، امسح النقطة واعد كتابتها حتى تصل الى code completion) يقوم Delphi باعطاء قائمة لكل الخصائص و onstructors و procedures و functions لهذا النوع من الـ object's. فى هذه الحالة، نود ان نعين String جديد لخاصية الـ Caption الخاصة بالـ object أبحث عن كلمة Caption فى القائمة، واضغطها مرتين لإضافة الـ Caption فى الـ source code. أكتب String جديد لتحديده للـ Caption فى الزر. على سبيل المثال، قم بكتابة هذا السطر ليكون كالآتى:

```
Button1.Caption := 'Ouch!';
```

٤- يمكنك الآن تشغيل البرنامج وضغط الزر، مما يغير نصه الى "Ouch!" .

Tip

فكرة: لتغيير كم الوقت الذى عليك انتظاره حتى تظهر نافذة Code Insight page tab، استخدم الـ Tools|Environment Options......، اختر Code Insight page tab واسحب Delay slider لتحديد القيمة التى تريدها . و فى desktop system فإننى استخدم اسرع توقيت وهى نصف ثانية . اما على desktop system ، فإننى ازيد هذه القيمة فيتظهر النافذة اذا انتظرت ثانية أو أكثر بهذه الطريقة، يمكننى الاستمرار فى الكتابة اذا فضلت عدم رؤية النافذة، والتى تأخذ لحظة متى تتشكل . جرب مواضع أخرى لترى أى توقيت يعمل بشكل افضل على نظامك .

وطريقة اخرى لاستخدام code completion هى ان تضغط قضيب المسافة+Ctrl . هذا يجبر الـ code completion على العمل، والتى تعطى قائمة بالإشياء المتاحة لهذا الموضع فى source code . وعلى النقيض من documentation التى قد تقرأها، هذه الخاصية لا تعطى قائمة بالقيم الصحيحة لتعيين عبارات - إنها تقوم قائمة بكل الاشياء الموجودة ولذا يمكن تخصيصها أو استخدامها بفاعلية فى هذا المكان فى البرامج . وقد يفيد هذا فى عبارات التعيين، ولكنه مفيد أيضاً فى جوانب أخرى، مثل إيجاد اسم glabal declaration .

على سبيل المثال، اننى اجد هذا النوع الثانى من الـ code completion غاية فى الافادة فى الاختيار من بين الـ local and global variable المتعددة للبرنامج لبدء عبارة تعيين، فإننى اضغط قضيب المسافة +Ctrl ثم اختار اسم متغير من القائمة الناتجة . وهذا يمكننى من استخدام اسماء متغيرات وصفية وطويلة مثل CountOfAliveObjects، واختيارها من القائمة بدلاً من كتابة هذه الاسماء مراراً .

Note

ملحوظة: تذكر انه بامكانك استخدام الـ code completion بطريقتين . الاولى، اكتب اسم object ثم ضع نقطة، ثم انتظر لحظة لتظهر قائمة بخصائص الـ object و methods . الثانية، اضغط قضيب المسافة +Ctrl فى أى وقت للحصول على قائمة بالـ objects المتاحة فى هذا المكان source code .

ان الـ code completion يتطلب ان تكون source code صالحة من ناحية الـ Syntaxs على الاقل فى مكان الادخال. اذا تلقيت رسالة "غير قادر على استدعاء الـ code completion" نتيجة اخطاء فى الـ source code ، تأكد من انك لم تنس كلمة اساسية أو أى عنصر هام آخر. لرؤية هذا الخطأ، حرك المؤشر بعد نهاية unit's source code (بعد نقطة النهاية) واضغط قضيب المسافة +Ctrl. فى الحقيقة قد لا يكون هناك اية اخطاء فى هذه الحالة - المسافة الواقعة بعد نهاية الـ unit تعتبر خارج نطاقcode، وهذا هو سبب تلقيك هذه الرسالة).

امامك طريقتين لتنظيم الـ code completion يمكنك تغيير الوقت الذى تستغرقه نافذة completion للظهور، أو يمكنك إيقاف code completion استخدم الـ.....Tools|Environment Options واختر الـ Code Insight لعمل أى من التعديلين قد تريد إيقاف code completion على الحاسبات البطيئة للغاية، والتى قد تستغرق وقت طويل لإظهار نافذة completion . فى هذه الحالة، فعليك ان تستخدم online help و documentation لترى الخصائص و methods و objects.

:Code parameters

واليوم الثالث من الـ code insight. جعل من المستحيل يرفض ما قد تحدده كقيمة بالنسبة لك methods الخاصة بالـ object وهى احد اخطاء البرمجة الأكثر شيوعاً. مع هذه الخاصية، يقدم نافذة قائمة بكل القيم المتغيرة المطلوبة بالنسبة للـ method التى يستدعيها code. فانت ببساطة تستعين بهذه المعلومات لتحديد القيم الصحيحة للـ parameters الخاصة بكل من الـ parameters والـfunctions.

التجربة ستظهر لك كم هى قيمة الـ code parameter هذه. واننى استخدم ايضاً code parameters لتوضيح كيفية استخدم هذين النوعين من code الـinsights معاً، وهو ما سنفعله دائماً. اتبع هذه الخطوات:

١ - ابدأ تطبيق جديد وأضف Button object على الـ form اضغط مرتين الزر لإنشاء الـ event handler.

٢- بين كلمة البداية الرئيسية وكلية النهاية الرئيسية، اكتب. Button1 (لاتنسى ان تضع نقطة) وانتظر لحظة. اكتب sc لإظهار ScaleBy method فى

نافذة code parameters، واضغط Enter لإضافة هذا method داخل source code (تذكر هذه الطريقة - عندما تظهر نافذة code parameters، اكتب الاحرف الأولى من method أو الخاصة التي تريدها، ثم اضغط Enter لإدخالها فى البرنامج.

٣- اضف اقواس الى عبارة الـ Button1.ScaleBy التى تم تشكيلها جزئياً ثم انتظر لحظة يجب ان ترى القائمة التالية من انواع الـparameters يمكن تحديدها للـ ScaleBy method.

M: Integer; D: Integer

يظهر هذا النص فى مربع تحت مؤشر النص مباشرة. لاحظ ان الـparameter الأولى مكتوبة بخط سميك - هذا يشير الى parameter الذى من المتوقع ان تكتب بعد ذلك. ادخل 4 وفصله، ومرة أخرى، انتظر لحظة. والآن يصبح D parameter مكتوبة بخط سميك مشيراً الى ان هناك قيمة رقمية متوقعة لهذا الـ parameter. اكتب فى، اغلق القرص ثم ضع فصلة منقوطة لإنهاء العبارة.

عندما تضع الاقواس، تختفى النافذة الخاصة بقيم الـ parameter، مما يشير الى انك قد قدمت كل الارقام المطلوبة. فكرة: تذكر ان تضع فصلة، وليس فصلة منقوطة، بين القيم الرقمية الحقيقية - وهى معلومة يعرفها واضعى برامج الـ Pascal المتمرسين ولكن يجهلها المبتدءين. والعبارة التامة يجب ان تبدو كالتالى:

```
Button1.ScaleBy(4, 3);
```

٤- يمكنك الآن ان تضغط F9 وتشغل البرنامج. ان الزر يكبر اكثر فى كل مرة تضغط عليه، وكأنه احد الاشياء الموجودة فى اليس فى بلاد العجائب. وهو يصبح هكذا لان ScaleBy method يحدد الـ VCL object بـ M مقسومة على D، وفى هذه الحالة 4/3 (أو ٣/٤) أو حوالى ١,٣٣ من حجمه.

Tip

فكرة: ان النافذة الخاصة لقيم الـ parameters قد تستغرق دقيقة أو اثنتين لتظهر، خاصة للـ parameters التى لها أكثر من arguments مختلفة كما هو الحال فى الـcode completion، يمكنك تغيير وقت التأخير لهذا الخيار أو يمكنك ابطاله بواسطة امر الـ Tools|Environment Options... .

:Tool-tip expression evaluation

ان code insight هذا يبدو أكثر غرابة عما هو فى حقيقته، ولكن لا تستطيع القول بأنه ليس فائدة. فى الواقع، قد تكون هذه الخاصية هى التى ستستخدمها فى معظم الاوقات فى اختيار الـ code الخاص بك. لاستخدام tool-tip expression evaluation، يجب ان تقوم بتشغيل برنامجك فى حالة debug ويمكنك فعل هذا باكثر من وسيلة، ولكن اشهر وسيلتين هما الـ breakpoint أو بتشغيل code فى نمط تنفيذ الـ code خطوة خطوة بضغط F8.

عندما تفعل هذا ويتوقف البرنامج فى مكان ما اثناء تنفيذه، حرك المؤشر فوق أى متغير لترى نافذة توضح قيمته. ويعتبر هذا ذو منفعة عظيمة خاصة فى فحص المتغيرات فى حالة debugging. ويجب ان يكون المتغير ظاهراً فى هذه النقطة من تنفيذ البرنامج ان المتغيرات التى تتصف بصفة الـ local بالنسبة للـ methodsوالتى يتوقف عندها البرنامج تعتبر غير قابلة للفحص (فى الحقيقة) هذه المتغيرات غير موجودة فى الذاكرة). وكمثال على كيفية عمل هذه الخاصية، اتبع الخطوات التالية :

١ - قم بانشاء تطبيق جديد، ادخل Button Button، اضغطه مرتين، وادخل العبارة التالية. فى Button1Click method :

```
Button1.Caption := 'Ouch!';
```

٢- إنها نفس العبارة التى قمت بكتابتها لمثال code completion فى هذا الفصل. قبل تشغيل التطبيق، ادخل breakpoint على العبارة السابقة. قم بهذا بالضغط الى اليسار من الخط، أو باستخدام امر الـ Run|Add Breakpoint.... .

٣- اضغط F9 لتشغيل التطبيق ثم اضغط الزر هذا يؤدى الى توقف البرنامج عند العبارة التى حددت بها الـ breakpoint. ولان البرنامج قد توقف فى حالة debug، يمكنك الان استخدام tool-tip expression evaluation لاختبار المتغيرات. حرك المؤشر فوق كلمة Caption . انتظر لحظة، ويقوم Delphi بعرض القيمة التى تم تعيينها حالياً لهذه الخاصة. وفى مربع صغير ترى الآتى :

```
Button1.Caption = 'Button1';
```

٤ - قد تم تحديد الـ Caption لهذا الـ String لان البرنامج لم يتم بعد بتنفيذ العبارة فى الـ breakpoint ، قم بتحريك المؤشر فوق الـ Caption ، والذى يظهر الآن القيمة الجديدة التى تم تعيينها فى آخر مرة تم فيها تنفيذ هذه العبارة .

كما هو الحال فى code completion و code parameters ، يمكنك تغيير وقت التأخير الخاص tool-tip expression evaluation ، أو إيقاف هذه الخاصية باستخدام امر الـ Tools|Environment Options . على الرغم من ذلك ، لا اعتقد انك تريد إيقاف مثل هذه الخاصية المفيدة .

Tip

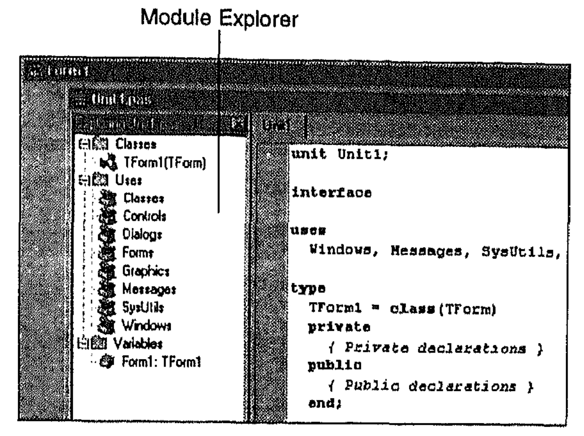
فكرة: Tool-tip evaluation تعمل بصورة افضل اذا تم إغلاق الـ compiler optimizations ، لأنه عند تشغيل الـ optimization ، قد يقوم الـ compiler باعادة ترتيب الـ code ، بطريقة ما بحيث يصبح وضع المتغيرات فى الذاكرة امر صعب أو مستحيل لـ debugger . ولإيقاف الـ optimization عليك اختيار الـ Project|Options... واختر Compiler page tab . وتحت باب Code generation ، امسح مربع خيار الـ Optimization . ويجعل هذا يحدث تلقائياً ، قم بتشغيل مربع خيار الـ Default اسفل نافذة dialog قبل ان تضغط OK . تأكد من تشغيل الـ optimization قبل إجراء عملية compiling النهائية .

: Module Explorer

ان Module Explorer ، وهو جديد فى Delphi 4 ، يبسط لك الطريق امام الـ source code files . ان الـ Module Explorer يقدم outline خاص بكافة عناصر برنامجك . ان هذا الـ object يعمل على document the source ويساعدك ايضاً فى ايجاد declarations محددة . يوضح الشكل (١-٤) الـ Module Explorer من شكل (١-١) .

يظهر الـ Module Explorer الى اليسار من code editor . وهو يعرض tree outline ، خاص بكل من classes ، properties ، methods ، global variables ، و global routines .

ويقدم الـ Module Explorer إيضاً قائمة باسماء الـ units المستخدمة من قبل الـ module .



شكل (١-٤): الـ Module Explorer

للربط بين الـ Module Explorer و code editor windows ، اضـغط Ctrl+Shift+E . قم بإخـتـيـار أى عنصـر فى tree-outline object ، ويقـوم Delphi بالإنتقـال مباشـرة الى مكان تعـريف الرمز فى code editor اننى اجد ان هذا مفيد جداً بالنسبة لتحديد موضع تنفيذ methods يكون قد تم تعريفه فى الجزء interface على سبيل المثال تعريف method من خلال class .

انModule Explorer يسـاعـد على البـحث بصـورة تصـاعـدية اكـتب الاحـرف الأولى من الرمـز الذى تريد ايجاده. اذا كـان هذا الرمـز معرف من خـلال البرنامج ، فإن المؤشر يذهب تلقائياً الى مكان فىcode editor .

: Docking and undocking

يمكنك إجـراء عـمليـتى Docking و docking لـ Module Explorer .
لإجراء عملية undocking اضغط واسحب الحد العلوى للنافذة . والذى يمكنك ان تسحبه الى أى موضع . اطلق زر الفأرة لعرض الـ Module Explorer فى نافذة منفصلة . وقد تقوم بتنفيذ هذه الحركة فى low-resolution display ، فى هذه الحالة ، قد يجعل الـ Module Explorer المعيارى الـ code editor صغيراً جداً . وفى حاسبى الذى يعرض ١٠٢٤×٧٦٨ ، افضل ان اترك الـ Module Explorer متصلاً بموضعه الاصلى

بعملية الـ undocking يمكنك اظهار نافذة الـ Module Explorer باختيار View|Module Explorer . إضـغط الـ Ctrl+Shift+E عند تعـاملك مع الـcode editor، هذا سوف يؤدى الى جلب الـ Module Explorer امام النوافذ الأخرى. ولأجراء عملية Docking لـ Module Explorer وإرجاعه الى مكانه الاصلى مرة أخرى، اضغط واسحب حد النافذة الى اليسار منcode editor . استمر فى السحب حتى يتغير حجم الـ outline من الحجم الحالى الى الحجم الأقل أترك زر الفأرة حتى تمام عملية الـ docking .

يمكنك إجراء عملية الـ Docking لـ Module Explorer الى اليمين من نافذة الـ code editor تحكم بالحد الأعلى الخاص به بواسطة الفأرة واسحب نافذة الـ Module Explorer الى جانب الآخر من editor . استمر فى السحب حتى يتغير حد حجم الـ outline من الحجم الحالى الى الأقل حجماً ثم اترك زر الفأرة.

Tip

فكرة: الـ code editor عندما يكون maximumized فقد يقل حجم الـ outline ويمكنك ترك زر الفأرة فى هذا الوقت لاستكمال عملية للـ docking .

Class Completion:

اذا قمت بالبرمجة مع Delphi ، فإنك من المحتمل إرتكابك للعديد من الاخطاء فى تعريف خاص بالـ method لأى class أو عند محاولة كتابتها بصورة صحيحة.

عندما تفعل هذا ، فتكون بالتأكيد قد نسيت ان تصنع اسم الـ class. والنقطة التى تليه ، أو كررت التعريف مرة أخرى ، يسبب تلك الأخطاء قد تظهر عدة رسائل توضح وجود خطأ منهما.

وباستخدام خاصية Module Explorer's class للـ completion والتى قد تعتبرها نوع من اداة من أدوات الـ code insight الجديدة - فإن code editor يمكنه إنشاء method shell لأى تعريف جديد. ولتجربة هذه الخاصية ، ابدأ تطبيق جديد، ثم اتبع هذه الخطوات :

١- أضف Button object على الـ main form الرئيسية للبرنامج.

٢- اضغط مرتين Button1 لإنشاء OnClick event الخاص للـ objects.

٣- اضـغط بالفـأرة TForm1 فى الـ Module Explorer . هذا يؤدى الى تحريك المؤشر الى تعريف TForm1 class. حدد موضع الكلمة الاساسية الخاصة وتحـتـهـا مـبـاشـرة اكـتـب تعـريـف procedure التـالى، والذى يوضح ان الـ Subroutine كعضو فى TForm1 class:

```
procedure Subroutine;
```

٤- لتكملة الـ code، عليك ان تحـــدد الـ implementation التى تتبع لها الـ method. ضع المؤشـر فى أى مكان من الجـزء الخـاص بتـعـريف TForm1 class، ثم اضغط Ctrl+Shift+C . يقوم Delphi بانتاج قائمة بكافة العناصر التى تكون معرفة دون التعامل معها.

٥- ومع الـ one unimplemented methad فـإنه يتم وضع المؤشـر بين البداية الرئيسية للـ Subroutine والنهاية الرئيسية لها. أضـف العبـارتان لاتمام الـ method كما يلى:

```
procedure TForm1.Subroutine;
begin
  ShowMessage('Ready to exit');
     Close;
end;
```

٦- حدد موضع OnClick event handler من الخطوة رقم ٢. لعمل هذا بسـرعـة باسـتـخـدام الـ Module Explorer، اضـغط Button1Click under TForm1's methods.

٧- اكتب العبارة التالية بين كلمة البداية الرئيسية وكلمة النهاية الرئيسية والاسلوب التام يجب ان يبدو كالاتى:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Subroutine;
end;
```

٨- اضغط F9 لتشغيل البرنامج. عندما تضغط الزر، يقوم OnClick event handler بإستدعاء الـ Subroutine، والذى يعرض رسالة وينتهى البرنامج.

ويعمل الـ Class completion ايضاً بشكل عكسى. قم بإضافة procedure أو function فى الجزء implementation section قم بتحريك المؤشر الى أى مكان داخل الـ procedure والـ function ، ثم اضغط Ctrl+Shift+C لإنشاء تعريف لهذا الـmethod فى الـ class المشار إليها. لتجربة هذا، عليك ان تمحو تعريف الـ class من القطاع الـ private للـ TForm1 . استخدم الـ Module Explorer للوصول الى تنفيذ الـ Subroutine واضغط Crtl+Shift+C. اذا لم يظهر أى تعريف للـ method ، يقوم Delphi بإضافة واحداً فى القطاع الـ private للـ calss . (يمكنك نقل هذا التعريف الى أحدى الجزئين الآخريين الـ protected أو public اذا لزم الأمر.

: Module navigation

ويقدم الـ Module Explorer طريقة أخرى لربط الـ code editor بين class declaration من خلال الـ class التابع له وبقية عناصره. ومع وضع المؤشر على أى method declaration اضغط سهم الـ Crtl+Shift+Down أو سهم الـ Crtl+Shift+Up للقفز الى الـ method's implementation (ليس من المهم أى مفتاح يهم سوف تضغط لان الخاصية هى الربط بين شيئين، والاتجاه لا يهم).

وإحدى الملحوظات فى ملف الخاصhelp بـ Delphi تشير الى انه بامكانك استخدام هذه المفاتيح للربط بين الجزء interface والجزء implementation فى الـ unit، ولكن هذا لا يبدو لى فعالاً. قد يكون هذا ثابتاً فى نسخة Delphi الخاصة بك.

الملفات وامتدادات اسم الملف:

ان الفقرات التالية تضيف محتويات انواع عديدة من الملفات التى يقوم Delphi بإنشاءها للتطبيق. وسوف اوضح لك أى الملفات يمكنك حذفها بسلام من دليل المشروع لتوفير مساحة من القرص.

ملفات Source code :

عندما تقوم بإضافة component على الـ form، وعندما تقوم بتعديل الـ event والخصائص للـ form، أو component، فإن Delphi يكتب نص برنامجك بالـ Object Pascal . وهذا النص يسمى source code البرنامج . فى أى وقت أثناء تطوير تطبيق ما، يمكنك استعراض ملفات الـ source code للبرنامج، ويمكنك طباعتها كنسخ احتياطية من الورق .

ان واضعوا برامج الـ Pascal ذو الخبرة قد يريدون تعديل الـ source code الخاصة بتطبيقاتهم . بغض النظر عن خبرتك، عليك ان تقوم بإدخال برمجة جديدة بحذر شديد .

ولكنك لا تستطيع فهم كل تغيير تقوم به . وللحصول على افضل نتائج، اتبع المقترحات الموجودة فى هذا الكتاب لإنشاء التطبيقات . وبمرور الوقت، سوف تصبح القواعد الخاصة بما تستطيع تغييره وما لاتستطيع تغييره واضحة لديك .

اتبع هذه الخطوات لاختيار source code لتطبيق مثال Hello ولتعتاد اكثر على البرمجة والملفات التى يولدها Delphi :

١- اذا لم يكن مشروع Hello مفتوحاً، اختر ....File|Open لفتح ملف مشروع Hello.dpr .

٢- اختر Select View|Units (Ctrl+F12) أو اضغط زر السرعة Select-unit-from-list، واختر Main from من قائمة modules . ويعرض هذا source code فى Delphi editor . اضغط Enter أو اضغط OK لإغلاق unit dialog وتوضح القائمة (١-٢) النص الذى تراه على الشاشة . ويتم تخزين نفس هذا النص على القرص المدمج على أنه Main.pas فى دليل الـ Source\Hello .

٣- اختر View|Units مرة أخرى ولكن هذه المرة اختر Hello module . وتظهر نافذة الـ editor الآن ملفين- اختر فيما بينهما بضغط الـ page tab فى أعلى النافذة . توضح القائمة (١-٣) نص ملف المشروع، الذى تم تخزينه على القرص المدمج على أنه Hello.dpr فى دليل الـ Source/Hello .

القائمة (١-٢): Hello\Main.pas

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
Forms, Dialogs,
  StdCtrls;

type
  TForm1 = class(TForm)
        CloseButton: TButton;
            procedure CloseButtonClick(Sender: TObject);
          private
        { Private declarations }
          public
        { Public declarations }
          end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.CloseButtonClick(Sender: TObject);
begin
  Close;
end;

end.
```

القائمة (١-٣): Hello\Hello.dpr

```
program Hello;

uses
  Forms,
    Main in 'Main.pas' {Form1};

{$R *.RES}

begin
    Application.Initialize;
    Application.CreateForm(TForm1, Form1);
    Application.Run;
end.
```

Tip **فكرة:** لإغلاق ملفات فردية فى الـ editor، تعامل مع هذه الصفحة واضغط Ctrl+F4، أو اضغط زر الفأرة الأيمن واختر Close Page. إن هذا ببساطة يغلق الصفحة فى editor- إنه لا يحذف فى الملف من القرص. ولإعادة فتح ملف source code خاص بـ unit ما، اختر اسمه باستخدام أمر الـ View|Units.

إن الـ source code لمشروع Delphi يتم تخزينها فى إثنين من ملفات النص على الأقل فى هذه الحالة، الـ Main.pas والـ Hello.dpr. ويعتبر ملف الـ Main.pas فى القائمة (١-٢) مثال لـ unit، والتى تحتوى على برمجة الـ form الرئيسية لبرنامج Hello وكل التطبيقات الموجودة فى هذا الكتاب لها على الأقل unit واحدة، ولكن المشروعات المعقدة من الممكن أن يكون لها العديد من الـ units كل منها قد تم تخزينه فى ملف pas. منفصل.

وملف second source الثانى فى تطبيق Delphi النموذجى يحتوى على source code لمشروع بأكمله. على سبيل المثال، يوضح Hello.dpr فى القائمة (١-٣) الـ source code لمشروع Hello. ولكل تطبيق ملف مشروع واحد فقط.

ولطباعة الملف الذى تم اختياره حالياً فى نافذة editor، اختر File|Print.... يمكنك الاختيار من بين العديد من الأوامر الموجودة فى Print Selection dialog

الناتج. على سبيل المثال، قم بتحديد خيار الـ Line numbers لإعطاء رقم لكل سطر برمجة، أو قم بتحديد خيار الـ Header/page number للحصول على المظهر التام. عليك بتشغيل خيار الطبع Syntax لطباعة الكلمات الرئيسية بخط سميك ولاستخدام الخط المائل underlining لعناصر البرامج الأخرى. قم بإيقاف طباعة الـ Syntax للحصول على طباعة سريعة على الطابعات البطيئة أو اذا كنت تفضل الحصول على مظهر واضح نسبياً في القوائم.

ملحوظة: إن النسخ الحديثة من Delphi تتضمن خياراً للطبع بالألوان. وتوضح القائمة الناتجة الكلمات الرئيسية وعناصر البرنامج الأخرى باستخدام ابراز تكوينات الجمل، وهو ما يشبه ما تراه في نافذة الـ Delphi editor. بالطبع، إنك تحتاج الى طابعة ألوان لاستخدام هذه الخاصية، وأنت أيضاً تحتاج الى استخدام خيارات الـ Tools|Environment Options... لإختيار ترتيب اللون البديل حسب نظام البرنامج- والمواضع الأخرى مثل Classic، على سبيل المثال، تبدو أفضل على الشاشة منها على الورق. قم بإجراء التجربة على ملفات إختيارية حتى تحصل على المنظر الذى تريده. للأسف، عليك أن تقوم بالفعل بطباعة صفحة لتغيير خيارات الطبع- لتغييرها وإلغاء إعادة الطبع الى المواضع القديمة.

ملفات الـUnit:

والآن، فلتنظر من قريب الى source code التى يولدها Delphi لمشروع ما، بداية من form unit الرئيسية، Main.pas [القائمة (١-٢)]. إن السطر الأول فى هذه الحالة هو الـ Main. ويعطى Delphi unit نفس الاسم الذى تحدده أنت عندما تقوم بحفظ unit file للمشروع.

ننتقل الى التعامل مع الجزء interface والتى تجعل العناصر التى تلى التعريف متاحة لوحدات أخرى ليستخدم هذا الاسم. ويحتوى جزء الـ interface على public declarations للـ unit. والعناصر التى ليست موجودة فى الجزء interface والتى يتم منع استخدامها مع العناصر الأخرى.

وغالباً ما تستخدم الـ units، عناصر unit أخرى. على سبيل المثال، إن Main تستخدم units أخرى عديدة، كلها موجودة فى قائمة تحت الجزء uses فى

أعلى الـ Unit على سبيل المثال Windows، Messages، SysUtils، وغيرها وهي توفر بيانات و code الخاص بـ Windows API، أو الاستعمال لبيانات وcode برنامج آخر، أو لـ Delphi component. وبإمكانك أيضاً إنشاء units خاصة بك واستخدامها. على سبيل المثال، بإمكانك إنشاء unit ذات بيانات وcode يمكن التشارك فيها مع أجزاء مختلفة من البرنامج.

في الـ Main.pas نأتي إلى قطاع type declaration، وعندما نضع كلمة تعريف العنصر فإننا نريد تحديد طبيعة العنصر الذي يمكن إنشاءه في الذاكرة. على سبيل المثال، إن الـ Integer يعد نوع بيانات يمثل الاعداد الصحيحة. ولاستخدام الـ Integer، عليك إنشاء متغير أو أكثر من هذا النوع يمكنه تخزين قيم معينة مثل ٣٢١. وتشمل أنواع بيانات الـ Pascal الأخرى characters، strings، records و الـ floating-point سوف أوضح الكثير عن هذه الأنواع وصادفت مثالاً عن هذا في الأمثلة لهذا الكتاب.

ونوع البيانات يمكن أن يكون أيضاً class بمعنى object-oriented structure يستخدم كثيراً في تطبيقات Delphi. وتصف الـ class خصائص الـ object الذي يمكن إنشاءه في الذاكرة. وقد تعرف class encapsulates. ويمكن للـ class أن تتعامل مع class members أخرى، على سبيل المثال، إن TForm1 class في الـ Main.pas تتعامل مع TForm1 class، وهي class يقدمها Delphi لإنشاء forms تسمى بالمشتق من الـ TForm1.

و TForm1 class في الـ Main.pas تعلن عن أثنان من الـ classes المضافان لي أولئك الذين ورثتهم في الـ member الأول هو TButton object، ويدعى CloseButton. والـ member الثاني من الـ TForm1 هو procedure، أو subroutine والذي يؤدي حدوث شيء ما. في هذه الحالة، ينهي CloseButtonClick procedure البرنامج عندما تضغط زر الـ Close الخاص بالبرنامج. ذلك هو الـ component و event hander الذي قمت بإنشاءه بإتباع الخطوات الموجودة في هذا الباب- Delphi يحدد الـ code الذي سوف يكتب وذلك عند إضافة object على الـ form وحسب صفات والـ event الخاصة لذلك الـ object.

ويمكن أن يكون للـ Class، عناصر توضيحية إضافية بعضها يمكن أن يكون خاص (يستخدم فى class unit فقط) أو عام (متاح أيضاً لعبارات فى units أخرى). هذه الحالة التى أمامنا ليس لها أى تعريف سواء أكان public أو private . ويشير التعليقان المحاطان بأقواس الى أين يمكن أن تذهب هذه الأنواع من الـ declarations .

```
private
  { Private declarations }
public
  { Public declarations }
```

إن التعليقات ليس لها تأثيرات فى وقت التشغيل . فهى مجرد ملحوظات تصف جانب من البرنامج . وقد يوضح التعليق عبارة محيرة، أو من الممكن أن يعرف مؤلف البرنامج، عدد مرات مراجعته، وحقائق أخرى مختلفة . يمكنك كتابة أى شئ تريده بين الأقواس . ويمكن أن تمتد التعليقات أيضاً لتشغيل أكثر من سطر واحد . وهنا مثال لتعليق متعدد السطور :

```
{ This multiline comment occupies three lines. It
begins with an opening brace on the first line,
and ends with a closing brace. }
```

وبدلاً من الأقواس، يمكنك استخدام الرموز ثنائية الحرف (*and*)- والمتبقيةمن نظم الـ Pascal عندما كانت كثير من الكلمات الرئيسية والختامية تفتقد علامة الأقواس . على سبيل المثال، يعتبر هذا تعليقاً مقبولاً للغاية :

```
(*Program author: Your name goes here*)
```

قد تخلط بين أنماط تعليق مختلفة فى نفس البرنامج . ولكن اذا بدأت تعليقاً بقوس، فعليك أن تغلقه بقوس وكذلك، اذا بدأت تعليق بـ *)، فعليك أن تنهيه بـ (* .

والنوع الثالث من التعليقات- والفريد فى الـ Object Pascal، وتم اقتراضه من ++C- يبدأ بشريطتين مائلتين ويمتد الى نهاية السطر الجارى . هذا النوع من التعليقات، لا يمكن أن يطول لأكثر من سطر واحد . على سبيل المثال، يمكنك استخدام هذا النوع من التعليق لإضافة ملحوظة توضيحية الى تعريف معين .

```
var
  Counter: Integer;  // Temporary loop counter
```

وأخيراً، يوجد فى قطاع interface الخاص للـ unit متغير أو أكثر يسبقه الكلمة الأساسية var . وقد لا يكون للوحدات أى متغير وقد يكون لها واحد أو أكثر . وهذا المثال له متغير واحد .

```
var
  Form1: TForm1;
```

ويقوم التعريف بإنشاء متغير يدعى Form1 من الـ TForm1 class . بمعنى آخر، يعتبر الـ Forml object class تحتل مساحة فى الذاكرة . والشكل فى هذا المثال يمثل النافذة الرئيسية لبرنامج Hello ، التى تحتوى على event handler يؤدى فعل الزر . مرة أخــــرى، يكتب Delphi هذا التـعريف عندما تقوم بإنشاء الـ form فليس عليك أن تقوم بهذا يدوياً .

ويتشابه الـ class object مع نوع البيانات مثل النوع Integer . يمكنك إنشاء واحد أو أكثر من الـ object لأى class ، تماماً كما يمكنك إنشاء متغيرات Integer متعددة .

ويأتى بعـد قطاع interface الخـاص بالـ unit قطاع الـ implementation حيث يقوم Delphi بإنشـاء عبارات لتنفيـذ run-time actions ، وحيث يمكنك إضافة التعريف وعناصر برمجة أخرى . يمكنك أيضاً كتابة عبارات داخل قطاع الـ implementation على سبيل المثـال، قم بتحديد موضع الـ procedure الذى كتبت قبله عبارة الـ ;Close . ها هى ذى مرة أخرى للإسترجاع :

```
procedure TForm1.CloseButtonClick(Sender: TObject);
begin
  Close;
end;
```

وقبل هذا procedure ، تجد هذا الأمر الذى قد يبدو غريباً :

```
{$R *.DFM}
```

وبالرغم من أنها تبدو كالتعليق، إلا أن علامة الدولار والواقعة بعد القوس المفتوح مباشرة تنبه Delphi الى أن هذا السطر هو أمر خاص يفتح ويقرأ ملف ينتهى بإمتداد اسم الملف dfm. (إختصار Delphi Form) . والنجمة تنبه Delphi الى أن يبحث عن ملف يحمل نفس اسم المشروع، ولكن ينتقى بإمتداد اسم الملف dfm. . ويحتوى هذا الملف على خصائص الـ form التى تم إدخالها مع الـ Object Inspector .

وفى آخر سطر من الـ Main.pas توجد كلمة النهاية الرئيسية يعقبها نقطة . من الناحية الفنية، تشير كلمة النهاية الرئيسية الى نهاية تركيبة الـ unit، وترمز النقطة الى نهاية الملف. ولكن، هذا يعد تمييزاً غير هام. فقط تذكر أن النهاية الأخيرة فى unit ما تنتهى بنقطة .

ملفات المشروع:

إن خبراء Pascal سوف يتعرفون على الـ Hello.dpr على أنه برنامج Pascal [راجع القائمة (١-٣)]. كما هو الحال فى Main unit، يقوم Delphi بإنشاء هذا النص تلقائياً. يعتبر مشروع Delphi بحق مشروع Pascal. إن للمشروع هدفان أساسيان توضيح الـ application unit وتنفيذ البرنامج .

إن أول سطر فى Hello.dpr يحدد اسم البرنامج فى تعريف البرنامج- فى هذه الحالة، Hello. فى السطر التالى، إنه يستخدم تعريف يحدد units أخرى يستخدمها البرنامج .

وهذه الحالة التى نحن أمامها تستخدم كلا الوحدتين. الأولى فى الـ unit Forms التى تقدم البرمجة اللازمة لإنشاء Delphi forms فى وقت التشغيل . الثانية هى Main unit فى الملف Main.pas. والسطر الثانى فى قطاع الـ uses بالنسبة للـ unit ويشير الى أن unit Main تم وضعها فى ملف Main.pas وإن Form1 هو اسم الـ form فى هذا الـ unit ويكتب Delphi كل هذا الـ code تلقائياً:

وأمر الـ {$R *.RES} . الذى يأتى بعد ذلك، يربط ملف الـ binary resource فى الملف الـ exe code. وتحتوى الـ resource file على ايقونة نظام البرنامج فقط، ولكن يمكن لها ان تحتوى على انواع اخرى من الـ resource. على

الرغم من ذلك، إنك لن تستخدم ملفات resource كثيراً في Delphi كما تستخدمها في نظم التطوير الأخرى مثل. الـ C والـ ++C.

اخيراً يوجد في المشروع ثلاث عبارات بين كلمة البداية الرئيسية begin وكلمة النهاية الرئيسية end، يتبعها نقطة تشير الى نهاية الملف. والعبارة الاولى تبدأ بـ Application object، والذي يوفر البيانات و code التي تنطبق على التطبيق بأسره. والعبارة الثانية تنشئ الـ form object للتطبيق، Form1 ، في الذاكرة والعبارة الثالثة والأخيرة تقوم بتشغيل التطبيق. (ويختلف هذا الـ code في النسخ الحديثة من Delphi – اذا كنت تستخدم نسخة قديمة من Delphi، فيمكنك ان ترى عبارتين فقط، ولكنهما يؤديان نفس الافعال المذكورة هنا).

Note **ملحوظة:** يقوم Delphi تلقائياً بإنشاء والحفاظ على الـ source code للملف مشروع الـ Hello.dpr. فنادراً ما تحتاج الى تعديل عبارات هذا الملف ولهذا السبب، فإن ابواب هذا الكتاب لا تضع قائمة التالية ملفات مشروعات التطبيقات ففي غالبية الحالات، تقوم بإضافة لغة البرمجة في unit modules التابعة لـ form معينة وتقوم بادخال هذه البرمجة في قطاع الـ implementation فقط، وليس في قطاع الـ interface.

امتدادات اسم الملف:

ان تطبيق Delphi النموذجي يحتوى على انواع مختلفة عديدة من الملفات. وينتهي كل اسم ملف بامتداد يعرف محتويات الملف. وتصف القائمة التالية الملفات وامتدادات اسم الملف الخاصة بها التي يقوم Delphi بإنشاها لمشروع Hello الخاص بهذا الباب، بالإضافة الى مشروعات قليلة اخرى. ولقد شرت ايضاً الى أي من الملفات التي يمكنك حذفها بأمان، في حالة اذا كنت تفتقر الى مساحة على القرص، أو – مثلي أنا – تقوم بالتطبيق وذلك بالتخلص من الملفات الهالكة.

ها هي الملفات التي ينشئها Delphi لمشروع Hello، مع بعض الملفات الأخرى التي قد تقابلها من حين الى آخرة:

● **ملفات ~*.:** والملفات التي بها اسماء تنتهي بـ ~*. (على سبيل المثال، Main.~dp) تعتبر نسخ احتياطية من الملفات التي تم تعديلها وحفظها. يمكنك ان

تحذف هذه الملفات في أي وقت وانت مطمئن، رغم انك قد تريد الاحتفاظ بها لتعويض الفقد أو التلف في البرمجة، أو للعودة الى مراجعة سابقة.

● **ملفات bpg.:** ملفات Borland Project Group ، والتي يمكنها تخزين مجموعات من المشروعات تعتبر بحق ملفات نص لأمر الـ Make وملف الـ Make يضع قائمة بـ modules التابعة ويوفر الأوامر والمعلومات اللازمة لعملية command-line لتطبيق متعدد الـ modules . والمبرمجون دون الخبرة فقط الذين يستخدمون أدوات command-line compilation .

● **ملفات الـ dcr.:** تحتوى ملفات الـ Delphi Component Resource على ايقونة Component كما تظهر على لوحة الـ VCL . إنك تستخدم ملفات dcr . فقط عند إقامة Component مخصصة . لا يجب ان تحذف هذه الملفات . اذا فعلت هذا، فإن Delphi يبدلها بايقونة ثم تخزينها للـ Component عندما تقوم بتركيبها في الـ palette .

● **ملفات الـ dcu.:** وتحتوى هذه الملفات على بيانات و code قد إجريت له عملية compiled على سبيل المثال، ويحتوى Main.dcu على الـ codes والبيانات في source code والتابع لـ Main.pas . يمكنك حذف ملفات الـ dcu . . الخاصة بالمشروع لان Delphi يعيد إنشائها عندما تقوم بعملية الـ compile مرة أخرى .

● **ملفات dfm.:** وتحتوى هذه الملفات على binary values تمثل خصائص الـ form وكذلك خصائص أي components يتم وضعها على الـ form والعلاقات بين الـ events والـ event handler يتم ايضاً تخزينها في ملفات الـ dfm . ويقوم Delphi بنسخ هذه المعلومات في ملف الـ exe . لا تقم ابداً بحذف ملفات الـ dfm . . وعليك معاملتها بنفس الحرص الذى تعامل به ملفات الـ source code .

● **ملفات dll.:** وتحتوى هذه الملفات على الـ code الخاص بـ dynamic link libraries . بالرغم من انه بامكانك حذف ملفات الـ dll. وانت مطمئن حيث انه لديك الـ source code لا تحذف ملف الـ dll. الا اذا كتبته أو كنت واثقاً من مصدره .

● **ملفات dof.:** وتقوم هذه الملفات بتخزين كافة عناصر المشروع . ويعتبر الـ dof هو إسم الإمتداد الخاص بـ Delphi Options File اذا كنت تستخدم أمر الـ

ProjectlOptionsلتغيير العناصر متنوعة لهذا المشروع، فإن Delphi يحفظ مواضعك في ملف dof. الخاص بالمشروع. والسبب الوحيد لحذف هذه الملفات هو للعودة الى العناصر الأساسية للمشروع. ويحل هذا النوع من الملفات محل ملف الـ opt. فى النسخ السابقة من Delph .

● **ملفات dpk.**: هذا النوع من الملفات يحتوى على package والتى تكون فى الغالب عبارة عن مجموعات من الـ units. على سبيل المثال، ان الـ VCL بأكمله يتوفر فى package. لا تحذف هذه الملفات ابداً. ان package فى الحقيقة عبارة عن ملف Pascal source code - فهو يضم code أجرى له عمليةcompiled والناتج files (dcu.). ولكنه لا يضم هو فى ذاته أى code .

● **ملفات dpr.**: وهو اختصار لـ Delphi Project. وتعتبر ملفات الـ dpr. فى الحقيقة ملفات Pascal source code فى معظم الحالات، يجب الا تقوم بتعديل ملفات dpr.، رغم ان واضعى البرامج الخبراء فى Pascal قد يفعلون هذا بحذر شديد. يقوم Delphi بإنشاء ملف dpr. عندما تقوم انت لاول مرة بحفظ مشروع تطبيق جديد. لا تحذف ابداً ملفات dpr..

● **ملفات dsk.**: تقوم هذه الملفات بتخزين مواصفات الـ application's desktop ولكن هذا فقط يحدث عند أختيارك للـ Autosave مع إختيارك للأختيار ToolslEnvironment. يمكنك حذف ملفات dsk. اذا لم تكن تريد الاحتفاظ بترتيب نوافذ الـ Delphi's، أو اذا اردت العودة الى التخطيط الأفتراضى حسب نظام البرنامج. رغم ذلك، تقوم ملفات dsk. بتخزين اسماء مسارات المشروع، حتى اذا ما حركت مشروعك الى مواضع أخرى، أو اذا ما قمت بتخزين modules فى دلائل متعددة، فيجب الا تحذف ملفات dsk.

● **ملفات exe.**: كما تعرف، ان الملفات المنتهية بـ exe. هى ملفات يطلق عليها executable code وكما ذكرت، يمكن ان يقوم Delphi بإنشاء exe code. file. وهو الوحيد الذى يجب عليك ان توزعه الى مستخدمى تطبيقك يمكنك باطمئنان حذف ملف exe. للمشروع لأن Delphi يعيد إنشاءه عندما تجرى عملية compile.

• **ملفات opt.**: راجع ملفات الـ dof.، والتي تحل محل ملفات opt. في النسخ الحديثة من Delphi . ان ملف الـ opt. قد اختفى الآن. اذا وجدت واحداً - عند تحديث تطبيق قديم، مثلاً - يمكنك حذفه وانت مطمئن بعد فتح وحفظ المشروع مستخدماً نسخة احدث من Delphi.

• **ملفات pas.**: تحتوى هذه الملفات على الـ Pascal source code . وتطبيق Delphi النموذجي له ملف pas. واحد لكل unit تابعة لـ form معينة، رغم ان مطوري Pascal الخبراء قد يقومون بتخزين برمجة أخرى في ملفات الـ pas.. يمكنك عرض ملفات الـ pas. باستخدام الـ Delphi's code editor، أو بأي ASCII text editor . لا تحذف ملفات الـ pas..

• **ملفات res.**: الملفات ذات اسماء تنتهي بإمتداد الـ res. وتحتوي على binary resources مثل ايقونة نظام البرنامج وأي bitmaps أخرى، وتبدو ملفات الـ Resource تتشابه بشكل واضح مع برمجة الـ nonvisual Windows من خلال الـ Borland Pascal ، CC أو ++C عنها في الـ Delphi . في الـ Delphi، استخدم الـ Image Editor في قائمة الـ Tools لإنشاء وتعديل ملفات res.. لا تقم ابداً بأية تغييرات للملف الـ resource للمشروع، والذي له نفس اسم المشروع ولكن ينتهي بامتداد اسم الملف res. اذا فعلت هذا عندما يقوم Delphi باعادة انتاج هذا الملف، يتم فقد أي resource تكون قد أضفته إلى الأبد.

Caution

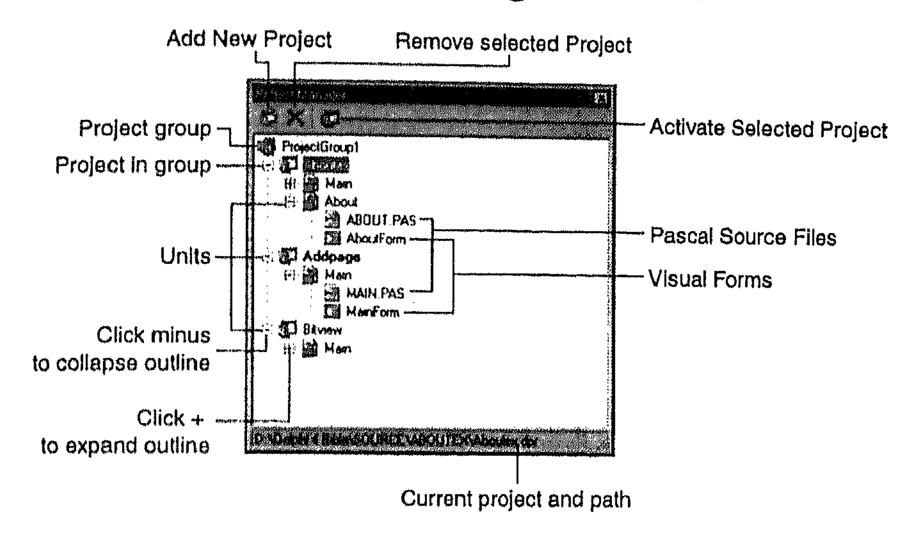
تحذير: لا تحذف ابداً ملفات لها اسماء تنتهي بـ dfm أو dpr. أو pas.، وتحتوي هذه الملفات على source code وخصائص التطبيق. وعند تدعيم تطبيق اثناء التطوير، فإن هذه هي الملفات الهامة التي يجب ان تحفظها.

:Project Manager

الـ Project Manager في Delphi 4 يعتبر اداة جديدة تماماً تحل محل النسخة السابقة، والذي اجري له تغيير بسيط من النسخة الأولى Version 1 والـ Project Manager لإنشاء ملفات Make لإجراء عملية compile من خلال command line بإستخدام الـ Make utility . ويستطيع الـ Project

Manager الجديد ان يجمع مشروعين أو أكثر مرتبطين، والذى يمكن فيما بعد إجراء عملية الـ compile لهما بأمر واحد.

يوضح الشكل (١-٥) نافذة الـProject Manager الجديد. يظهر الشكل ثلاث من المشروعات الموجودة على القرص المدمج الخاص بهذا الكتاب والتى، قمت أنا بتصنيفها فى مجموعة مشروع واحدة.



شكل (١-٥): نافذة الـProject Manager الجديد فى Delphi 4

استخدام الـProject Manager :

تعتبر غالبية أوامر الـProject Manager بديهية. اختر مشروعاً واضغط Activate Selected Project speed button يمكنك ان تضيف مشروعاً خالياً وجديداً بضغط Add New Project speed button. أو يمكنك حذف مشروع بضغط زر الـ Delete Project . يمكنك ايضاً اضافة وحذف ملفات فردية الى مشروع أو آخر.

تتغير اوامر pop-up menu الخاصة بالـ Project Manager اعتماداً على أى نوع من العناصر الذى تم إختياره على سبيل المثال، قم بإختيار اسم مشروع واضغط زر الفأرة الأيمن للحصول على قائمة بالأوامر التى يمكنك تطبيقها على البرنامج. قم بإختيار اسم ملف واضغط زر الفأرة الايمن للحصول على قائمة بالأوامر التى تنطبق على ملفات فردية.



وكثير من اوامر الـProject Manager تتوفر ايضاً في قائمة الـProject . وهذه الاوامر تتغير لتعكس المشروع الذي تم اختياره حالياً، والذي يتم عرضه بخط سميك. يمكنك فتح وإغلاق اية ملفات في أي مشروع معروض في نافذة الـ Project Manager ولا يجب ان تكون الملفات متعلقة ببعضها.

على سبيل المثال، باستخدام المعلومات الموجودة في الفقرة التالية حول إنشاء مجموعات مشروع، يمكنك فتح الـsource code unit في مشروعين منفصلين، لا علاقة لهما ببعض. ويعتبر هذا مريحاً للعمل على مشروعات متعددة والتي، قديماً، كان يجب عليك ان تقوم بفتحها وإغلاقها منفردة (أو تحميلها في نسخ متعددة من Delphi).

Tip **فكرة:** يمكنك توصيل نافذة الـProject Manager لجعلها اسهل في العثور عليها. اضغط واسحب حد النافذة الى اليسار أو اليمين من الـ code editor، وثم أترك زر الفأرة. إفصل النافذة بالضغط وسحب حدها الأعلى ثم اترك زر الفأرة.

إنشاء project groups:

يستطيع الـ Project Manager الجديد ان يجمع مشروعات متعددة، والذي قد يكون مفيداً لإجراء عملية الـ compile لأكثر من مشروع متصل ببعضه في وقت واحد. اتبع هذه الخطوات لإنشاء sample project group والتي استخدمتها لإعداد الصورة الموجودة في شكل (١-٥):

١- اختر File|New واختر New page tabs . اضغط مرتين ايقونة الـProject Group لإنشاء ملف Project Group جديد وافتح نافذة الـProject Manager .

٢- لإعطاء project group اسم مختلف (والأسم حسب نظام البديل الإفتراضي للبرنامج هو ProjectGroup1)، قم بحفظ المشروع في أي دليل. اذا كنت تجمع مشروعات موجودة في مجموعة، فقد تريد إنشاء دليل منفصل لملف المشروع، ولكن هذا يرجع لك. يمكنك تخزين المجموعة في أي مكان تريده. لحفظ الـ Project Group اخترها في نافذة الـProject Manager ، واضغط زر الفأرة الأيمن في النافذة. اختر ..... Select Save Project وأعط اسم ملف.

٣- افتح pop-up menu مرة أخرى (تأكد من ان بند الـ ProjectGroup1 قد تم اختياره). واختر أمر الـ Add Existing Project. تصفح أى دليل تطبيق لـ Delphi، وافتح ملف مشروع dpr. لإضافته الى المجموعة. يمكنك ايضاً استخدام امر الـ Add New Project لإنشاء مشروع جديد للمجموعة.

٤- احفظ المجموعة التى تم تعديلها. ولاستعراض الـ Source code الخاص بالـ project group ، افـتح مـرة أخـرى الـ pop-up menu واخـتـر الـ View Project Group Source. توضـح القـائـمـة (١-٤) source code للـ project group للمشروعات الموضحة فى شكل (١-٥). (هذه القائمة غير موجودة على القرص المدمج - فهى لا تخدم اية اهداف عملية وموضحة هنا للعرض فقط). كما ترى، يعتبر ملف الـproject group فى الحقيقية ملف نص Make، والذى يمكن استخدامه مع الـ Make utility لإجراء عملية compile لأكثر من مشروع بإستخدام الـ Delphi command-line compiler.

القائمة (١-٤): ملف project group هو فى الحقيقية عبارة عن command-line Make file

```
#-------------------------------------------------------------
VERSION = BWS.01
#-------------------------------------------------------------
!ifndef ROOT
ROOT = $(MAKEDIR)\..
!endif
#-------------------------------------------------------------
MAKE = $(ROOT)\bin\make.exe -$(MAKEFLAGS) -f$**
DCC = $(ROOT)\bin\dcc32.exe $**
BRCC = $(ROOT)\bin\brcc32.exe $**
#-------------------------------------------------------------
PROJECTS = Aboutex Addpage Bitview
#-------------------------------------------------------------
```

```
default: $(PROJECTS)
#-------------------------------------------------------
Aboutex: ..\Delphi 4
Bible\SOURCE\ABOUTEX\Aboutex.dpr
 $(DCC)
Addpage: ..\Delphi 4
Bible\SOURCE\ADDPAGE\Addpage.dpr
 $(DCC)
Bitview: ..\Delphi 4
Bible\SOURCE\BITVIEW\Bitview.dpr
 $(DCC)
```

ولن اخــوض فى الاوامــر الموجــودة فى ملف الـ project group - فــإن المطورين ذوى الخبـرة فـقط هم الذين يريدون فـهـمـهـا . على الرغم من ذلك ، فـإن اهمية المجموعة هى انها تستطيع تبسيط عـملية تصفح إجراءات عـملية الـ compile لأكثر من مشروع وهذه الخاصية للـ Project Manager الجديد تجعله مثيراً لفريق من المبرمجين الذين يعملون فى إعداد أكثر من مشروع .

ويمكن لمجموعة واحدة ان توفر الوصول الى كل الـ source code الخاص للتطبيق من داخل دورة واحدة لـ Delphi .

افكار للمستخدم الخبير

● فى نسخ Delphi القديمة وبخاصة اذا لازالت تستخدم Windows 3.1 ، اذا تلقيت رسالة خطأ تقول "out of resources" ، خلال تركيب Delphi ، فإنه غالبا يكون أحد أمرين إما out-of-date , أو nonstandard video driver . استخدم الـ Windows Setup لتغيير الـ driver الى Standard VGA . يمكنك هذا من تركيب الـ Delphi . اتصل ببائع الـ VGA الذى تتعامل معه لتحصل على driver حديث . فلا يجب ان يحدث هذا مع النسخ الاحدث لـ Delphi .

● يصـعب عليك التعـامل مع الـ form و editor windows حيث أنهـما غالباً ما يخفيان بعضـهما البعض ويصعب إيجادهما . وللانتقال بينهما بطريقة اسهل، اضغط F12 و F11 . جرب هذا الآن لترى ما تفعله هذه المفاتيح .

● ان hint boxes الخاصة بـ Delphi، والتي تظهر عندما تضع المؤشر على ايقونة الـ component أو الـ speed button التابعة للـ form التابعة له . ولا تظهر نفس الـ Hint boxes اذا كنت تتعامل مع تطبيق آخر .

على سـبـيل المثـال، اذا قـمت بالتـعـامل مع Start menu bar في الـ Windows 95، فإن Delphi hint boxes تختفي فإذا لن ترى hint boxes بعد الانتظار لمدة معقـولة، اضغط مـؤشر الفـأرة على Delphi's title bar أو على أي نافذة أخرى مثل الـ Object Inspector، ثم حاول مرة أخرى .

● للحصول على تقرير مفصل اثناء عملية الـ compile والـlinki اختر أمر الـ Tools|Environment Options.....، اخـتـر Preferences page tab وقم بعرض التقرير المفصل . عندما تقوم بعد ذلك بعمل compile، link لبرنامج ما يعرض الـ Delphi، تقرير للحالة والذي يوضح الملف وعدد السطور التي حدث لها عملية compile، link وهذه المعلومة لا تعد ذات قيمة للغالبية المطورين، الا انهاء تؤكد ان شيئاً ما يحدث خلال عملية compile . كن على حذر : ان عرض displays يستقطع وقتاً من عملية compilation، واذا قمت بكتابة كثير من التطبيقات الصغيرة، فقد تأخذ وقتاً اطول لعرض الـ dialog بالنسبة إلى الوقت اللازم لإجراء عملية compile لهذه البرامج .

● اضغط مرتين على الـ form أو code editor ليزداد حجمها حتى يتنسابان مع حجم الشاشة . واذا كان هذا يخفي ازرار وقوائم Delphi، اضغط Alt لتجعل toolbars و Delphi's component palette تظهر مرة أخرى يمكنك ايضاً ان تفعل هذا بضغط Alt+F +Alt- على سبيل المثال- لفتح قائمة الـ File . اضغط Alt، اذا لزم الأمـر، للعـودة الى expanded window وطبـيـعي يظهـر الـ Delphi الـ expanded editor تحت VCL palette ، لزيادة حجم نافذة لتغطى كل الشاشة، اختر أمر الـ Tools|Environment Options، واختر display page Display tab (سابقاEditor)، واغلق الـ Zoom الى مساحة شاشة الكاملة .

• يمكنك استخدام الـ editor window لعرض أى Pascal source code file (والذى ينتهى غالباً بامتداد اسم الملف pas.) أو ملف نص (غالباً ما ينتهى بـ txt.). اختر أمر الـ ....File|Open أو اضغط Open-file speed button. وللاستفادة من ابراز الجمل الخاص بـ Delphi، يجب ان ينتهى الملف بامتداد اسم الملف الذى تم ادخاله فى مربع نص الـ ....Tools|Environment Options Editor Syntax-extensions. وكل امتداد يكون مفصول بفصلة منقوطة.

• وطبقاً للعرف، ان العبارات الواقعة بين كلمة البداية الرئيسية وكلمة النهاية الرئيسية - وتسمى من الناحية الفنية block- تحدها مسافتان [راجع القائمة (١-١) و (١-٢)].

وترك المسافات ليس مطلوباً، ولكنه يساعد على جعل البرنامج مفهوماً بتوضيح أى العبارات تتبع بعضها. وطبقاً للعرف، ان العبارات التى تحدها مسافات لها نفس المستوى تكون مرتبطة نوعاً ما.

• من خلال الـ display ٦٤٠×٤٨٠، اختر Tools|Environment|Options واختر Display page tab (سابقا Editor). حدد خط النص بـ Courier New (والبنط الأصلى) وحدد حجم بـ ٩ نقاط. وبهذا الوضع، والذى يعتبر مفيداً للـ laptops، يُعرض غالبية الـ source code فى الـeditor والذى يحدد حجمها حسب النظام الافتراضى للبرنامج دون ان يتطلب هذا ظهور الـ horizontal scrolling. اذا كانت عينيك افضل من عيناى، جرب حجم الثمان نقاط لترى مزيداً من النص. قد تريد ايضاً ان تجرب مع خيار الـ mappings الخاص بـ dialog's Keystroke، والذى يشكل أوامر لوحة مفاتيح الـ editor. اختر الـ Default اذا كنت مستخدم خبير فى الـ Windows، واذا كنت على علم بـ Borland Pascal أو الـ ++C، أو Brief أو Epsilon.

• لإختبار الـ multiple running applications، ابدأ من خلال الـ first instance فى الـ Delphi ثم استخدم الـWindows Explorer لبدء الثانى. أو، لإختيار instance الثانية أو instances المتعاقبة، ابدأ instances الأولى مع الـ Explorer ثم قم بتشغيل instance النهائية من Delphi. استخدام اوامر debugging فى قائمة الـ Run لإدخال عبارات البرمجة للـ instance التى بدأت مع Delphi.

● ان الخصائص والـ event relationships يتم تخزينها فى ملفات dfm. وهى ليست ملفات نص- إنها binary data file - ويجب الا تستخدم text editor لتعديلها. على الرغم من ذلك، اذا قمت بفتح ملف file. باستخدام FilelOpen........، يقوم Delphi بعرضه كانه نص. (جرب هذا الآن بفتح ملف Hello.dfm الخاص بمشروع Hello) وسوف يتعرف واضعى برامج Pascal على هذا النص على انه object-type declaration والتى تحدد خصائص وعلاقات الـevent للـcomponent . إنها ليست Pascal حقيقية، ولكن قريبة جداً ولكن فريدة فى الـ .Object Pascal. ولحفظ نسخة قابلة للطبع من هذه المعلومة، اختر FilelSave As....... واحد تسمية الملف بامتداد اسم ملف txt.. يمكنك ايضاً ان تفعل العكس - افتح ملف النص، قم بتعديله، واحفظه مع امتداد اسم الملف dfm. لتحويل النص مرة اخرى الى binary. والمبرمجون الخبراء قد يفعلون هذا، على سبيل المثال، لتعديل خصائص المشروع، رغم انه من الاسهل، فى غالبية الحالات، استخدام نافذة الـ Object Inspector لهذا الغرض.

المشروعات التى يمكنك تجربتها

(١-١) : افتح مشروع Hello واختر زر Close للـ form. قم بتغيير خاصية الـ Default لهذا الزر لتصبح True اضغط F9 لإجراء عملية compile، link وتشغيل البرنامج الذى تم تعديله. لانهاء البرنامج، يمكنك الآن ضغط Enter، والذى أختير على أنه الـ control الأساسى حسب النظام فى الـ Window. وعودة إلى الـ Delphi، حدد خاصية Cancel فى زر الـ Close بـ True. الآن يمكنك تشغيل البرنامج، يمكنك ايضاً ضغط Esc للإنهاء. ولكن غالباً ما يكون component واحد فقط فى الـ form يكون له خاصية Cancel أو Default الخاصة به محددة بـ True .

(١-٢): حاول إضافة components اخرى الى Hello أو إنشاء تطبيق اختيارى جديد. ان بعض الدقائق تقضيها فى اللعب قد تساعدك على ان تصبح اكثر اعتياداً على تخطيط Delphi.

(١-٣) : اختر أمر الـ Tools|Environment Options لتنظيم بيئة Delphi . استخدام Editor and Colors page tab لتعديل خصائص الـ editor، ولاختيار الوان ابراز تركيب الجمل . يستطيع Delphi عرض الكلمات الرئيسية وعناصر اخرى فى ملفات source code بنص مائل، وألوان مختلفة، وخط سميك وما الى ذلك. ان ابراز تركيبات الجمل ليس للعرض فقط- ان الاختيار الصائب للالوان وخصائص النص يساعد على جعل البرامج اكثر وضوحاً بابراز العبارات والكلمات الرئيسية والتعليقات والعناصر الأخرى .

(١-٤) : ولمزيد من الاطمئنان، اختر أمر الـ Tools|Environment Options، اضغط Preferences tab، وقم بتشغيل خيارى الـ Autosave . والخيار الأول، ملفات الـ Editor، يقوم تلقائياً بحفظ كل الملفات فى مشروع عندما تقوم بعملية الـ compile و ومن ثم تنفيذه، والخيار الثانى، Desktop، يحافظ على ترتيبات نوافذ Delphi. مع هذا الخيار، وعندماتعيد بدء Delphi، يمكنك أختيار ما سبق فى الدورة السابقة .

Tip

فكرة: اذا كنت تقوم بتشغيل كثير من البرامج التجريبية القصيرة لاختيار خصائص الـDelphhi، فقد تريد ايقاف خيار ملفات Autosave Editor لخفض نشاط القرص. وعند تطوير تطبيق كبير ذو ملفات عديدة مختلفة، فقد يمنع هذا الخيار الفقد الذى يحدث لملف هام بطريق الصدفة .

ملخص:

● يعتبر الـ Delphi نظام تطوير تطبيقات سريع، مناسب لإنشاء عينات نموذجية من الـ Windows و تطبيقات تامة تضاهى وتفوق فى السرعة والكفائة برامج كتبت فى الـ C ، C++ ، Borland Pascal 7.0 ، و Visual Basic، وكذا البرامج التى تم إنشاءها بوسائل اخرى ان الخصائص الجديدة فى الـ Delphi 4 تشمل امتدادات لغة الـ Object Pascal ، و Project Manager جديد، و Module Explorer للملفات الـ source code للـ unit . ومن الجديد فى

Delphi ايضـاً خـصـائص الـ debugging وتطور الـ VCL و client-server component وقاعدة البيانات .

● ويتكون تطبيق Delphi من واحدة أو أكثر من الـ form والتى تضع عليها الـVCL components ويمكنك تعديل خصائص الـ form و components مثل الـ Names و Captions الخاصة بهم، ويمكنك انشاء Pascal procedures لإداء events مثل استخدامك لأحد ازرار الفأرة .

● وتوفـــر الـ Code insights قـــوائم لـلـ parameters والـ code completion ويمكن ان تسـاعـدك فى توفيـر وقت الكتابة والوقت المنقـضى فى البحث فى online help screens و documentation وبالإضـافة الى خصائص التحرير المفيدةهذه، هذا بالإضافة الى ميزة tool-tip expression evaluation والذى يظهر قيمة المتغير فى حالة debug .

● عندما تقوم بتشغيل تطبيق، يقوم Delphi بعملية الـ compile و link للـ source code و modules الأخرى لإنشاء ملف exe. وهذا هو الملف الوحـيد الذى تحتاج ان توزعه على مستخدمى برنامجك .

● يمكن للـ Project Manager الجديد الخاص بـ Delphi ان ينشئ project group لمشروعات متعددة وهذا يبسط عملية المتصفح فى ملفات مشروعات متـعـددة، وينشئ ايضاً ملف Make لإجراء عـملية الـ compiling لمشروعات متـعـددة . وتمكنك خاصيـة الـ Project Manager الجديدة من تصفح أجزاء للcode .

فى البـاب التالى والمقـدمة لـ Visual Components ، سوف تعرف المزيد عن كيـفـيـة إضـافـة الـ component فى الـ application's form ويقـدم البـاب مجموعة الـ components فى الـ Delphi مع أمثلة من التطبيقات التى تظهر كثير من الخصائص والـ event الخاصة بالـ Visual Component .

*****

الباب الثانى
معلومات حول
الـ Visual Components

محتويات الباب:

- Visual Component Library
- Standard components
- Additional and Dialogs components
- System components
- Win32 components

إن البرمجة من خلال الـvisual components يعد أمر غاية فى السهولة. يمكنك أن تنجز الكثير بأقل مجهود.

والـ visual component، هو object مثل أى زر تقوم بإضافته على الـ form. فى الواقع، إن الـ form هى أيضاً object تستطيع أن تحمل objects أخرى. وكما تطور تطبيقات الـ Delphi، فسوف تقضى معظم وقتك فى إضافة، وتعديل visual component objects، لذا فمن المهم أن تفهمهما بتركيز. والـVisual components هى العناصر الأساسية التى ترتكز عليها أساسيات الـcode الخاص بك.

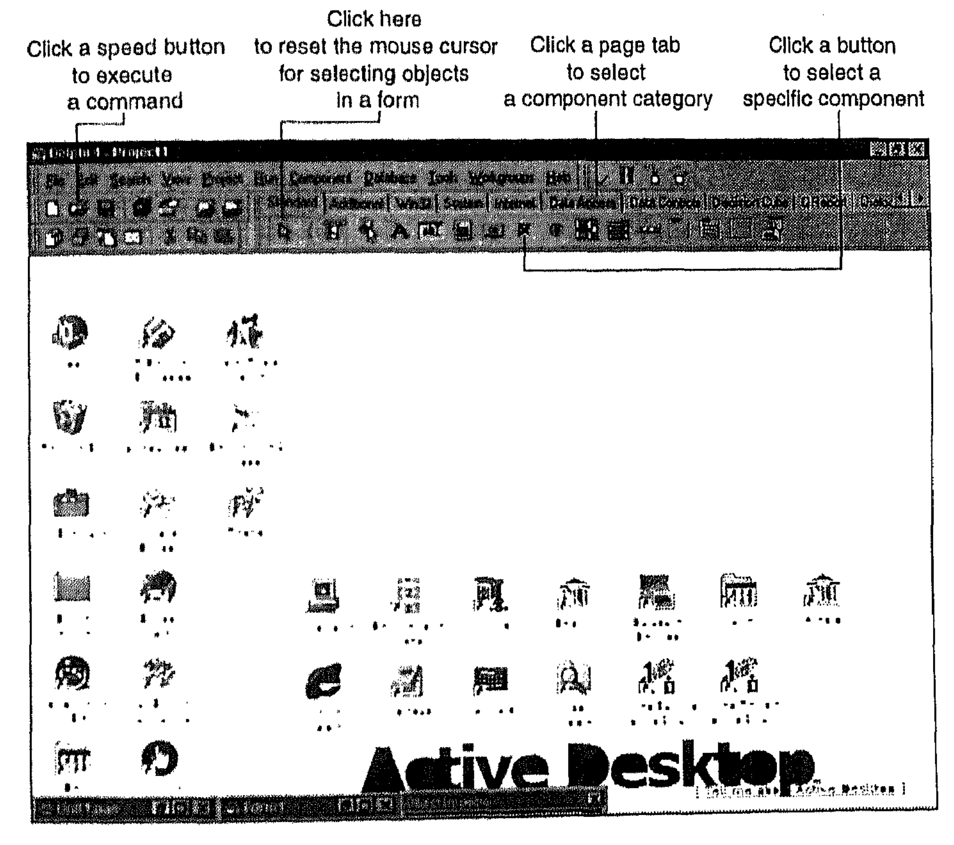
الـVisual Component Library:

مع Visual Component Library (VCL)، يمكنك إنشاء تطبيقات مفيدة مع جزء يسير من البرمجة. ولكن لا تأخذ كلامى ثقة؛ جرب أمثلة من التطبيقات الموجودة فى هذا الباب لعرض التطور السريع للتطبيق مع Delphi.

بعد التمكن من أساسيات الـ visual components فى هذا الباب، يمكنك أن تجد مزيداً من المعلومات عن component معينة فى أبواب الجزئين الثانى والثالث.

:Visual component categories

لإختيار component، اختر page tab مثل Dialogs فى الـ Delphi's component palette [أنظر شكل (٢-١)]. وفيما يلى قائمة بعناصر الـ Delphi visual component فى ترتيب ظهورها على الشاشة. إن عدد وترتيب الفئات على شاشتك قد يختلف إعتماداً على نسخة وطبعة الـ Delphi التى معك، وخيارات التركيب، ومكونات إضافية تكون قد ركبتها. والترتيب قد يختلف أيضاً اذا كنت قد استخدمت الـ Tools|Environment Options لترتيب أيقونات الـ palette.



شكل (٢-١): لوحة الـ visual component فى الـ Delphi وتظهر الـ page tab والتى تنظم الـ component فى فئات



- **Standard:** Standard Windows، مثل الأزرار، و labels، ومربعات الإدخال، والقوائم، و check boxes و scroll bars.

- **Additional:** Custom controls مثل bit buttons الذى لديها خاصية graphic images والتى تسمى (glyphs)، toolbar speed buttons، string، graphical grids، scroll boxes و graphics shapes و bitmap images و window splitters و check box lists و static text، و Simple chart tool.

- **Win32:** NT controls و Windows 95 ذات الـ ٣٢ بت القياسية بما فى ذلك page controls، image lists، rich-text editors، and track progress bars، animation windows، date-time picker و status bars و tree list view objects، و and toolbars.

- **System:** System service components وتشمل paint boxes, multimedia player و timers، Dynamic Data Exchange و OLE.

- **Internet:** Internet components و الـ World Wide Web وأدوات الـ Internet/Intranet protocol HTTP، HTML، FTP، connection objects مثل TCP و UDP.

- **Data Access:** الـ Components المخصصة للتعامل مع قواعد البيانات وذلك من خلال tables، SQL queries وبالإضافة أنك تستطيع مع هذه الأشياء إنشاء تقرير بواسطة QReport.

- **Data Controls:** Data-aware components للتعامل مع البيانات الناتجة عن قواعد البيانات. فاستخدام هذه الـ components والتى تشمل navigators، edit boxes، memos، pictures, check boxes و lists.

- **Decision Cube:** هذه الـ Components، تكون تقرير يطلق عليها decision reports وهى على سبيل المثال cross tabulations مع المعلومات الخاصة بقواعد البيانات- ولا توجد إلا فى نسخة الـ client-server.

● **QReport:** مـجـمـوعـة كـبـيـرة من الـ objects مـتـصلة بـ data-aware component . وهذه الـ component تحل محل تطبيق ReportSmith الذى كان مـتـوفـراً فى النسخ الأولى من Delphi . لمزيد من المعلومـات حـول QReport component، أنظر "Additional and Dialogs Components" فـى هـذا الباب وكذا الباب السابع عشر . أضف الى ذلك أن Delphi يقدم user manual من خــلال Microsoft Word لهـــذه component الكثـيــرة . أنظر ملف Qrpt2man.doc الموجود فى دليل Quickrpt الخاص بـ Delphi .

● **Dialogs:** مخصـصة هذه الـ component لأداء مهام معينة مثل إختيار الملفات والأدلة، إختيار الخطوط والألوان، والطباعة وإيجاد أو إحلال بيانات فى الوثائق.

● **Win3.1 :** وهذه الـ component تتطابق مع الـ components التى كـانت فى نسـخـة Delphi الأولى التى تعـمـل فى بيـئـة Windows 3.1 . وجـمـيـع الـ components فى هذه الفئة لها بدائل ذات ٣٢ بت . ولا يحتاج غالبية مبرمجى Delphi الى هذه الـ components ، وهذه الفئة يمكن حذفها من لوحة المكونات اذا أردت . لعـمل هذا، استـخـدم الـ .....Tools|Environment Options واضـغط Palette page tab.

● **Samples :** هناك components قـد تم وضع الـ source code الخـاص بهـــا فى Source\Samples الخـــاص بـ Delphi . والغــــرض من هذه الـcomponents هو فى الغـالب ان تكون مرشـدة لك فى إنشـاء الـ component الخاصة بك، وتجدها مفيدة لك فى التطبيقات ولم تتغير هذه الأمثلة كثيراً عبر تاريخ Delphi، ويمكن حذف هـذه الفئة وانت مطمئن فهناك component تعطى نفس الإمكانيات.

● **ActiveX :** تمثل الجزء الثـالث من الـ sample components وهى توضح استـخدام وبرمجة الـ ActiveX بإسـتخدام Delphi . لمزيد من المعلومـات حـول ActiveX ، انظر البـــاب العــشـــرين من هذا الكتـــاب، "بناء الـ Custom Components" .

• **MIDAS :** وهى مثل الـProvider، RemoteServer ClientDataSet, وغيرها لإنشاء تطبيقات الـ multitiered database وتتوفر هذه الخاصية فى النسخة client-server .

الأمثلة التطبيقية:

تقوم ثلاث أمثلة تطبيقية فى هذا الباب بأداء مهام نافعة اثناء عرض الاساليب التى تحتاجها فى برمجة Delphi . يظهر تطبيق الـ MemoPad التعامل مع الملف والقائمة . ويظهر تطبيق الـ BitView كيفية استخدام graphics and dialog boxes. ويظهر تطبيق الـ DClock كيفية استخدام Timer component لإنشاء عمليات تعمل أثناء تنفيذ عمليات أخرى .

إننى اقترح عليك إنشاء هذه التطبيقات بنفسك باتباع الإرشادات والمقترحات الموجودة فى هذا الباب- فهى طريقة عظيمة لتعليم تقنيات Delphi . لا تقم بتشغيلها - فلن تتعلم أى شئ بهذه الطريقة . بالطبع، ان القرص المدمج الخاص بهذا الكتاب يوفر جميع التطبيقات وملفات الـ source code الخاصة بها. انسخ دلائل المشروع المختارة من على القرص المدمج الى محرك القرص الصلب الخاص بك، ثم استخدام الـ ....File|Open لتحميل ملف dpr. الخاص بالمشروع. اضغط F9 لتنفيذ التطبيق. يمكنك ايضاً استخدام الملفات المتوفرة اذا تعثرت اثناء إنشاء البرامج الخاصة بك. انظر فقرتى "Win32 Components" و "Additional and Dialogs Components" فى هذا الباب للتعرف على بعض الـ component الأكثر تعقيداً فى Delphi .

On the CD-ROM

على القرص: جدول (٢-١) يضع قائمة بالإمثلة التطبيقية هذا الباب، والتى تم تخزينها على القرص المدمج المرفق بهذا الكتاب فى دلائل تتفق مع اسماء التطبيقات. على سبيل المثال، سوف تجد ملفات تطبيق الـ MemoPad فى دليل الـ MemoPad. ويوضح الجدول ايضاً فئات الـ components التى يستخدمها كل تطبيق.

Standard Components :

مع الـ Standard component، يمكنك إنشاء Windows objects وعناصر اساسية عامة مثل الازرار و check boxes والقوائم. لان الـ Standard

components سهلة الاستخدام نسبياً، فإنها تعتبر مقدمة جيدة لبرمجة الـ visual component.

جدول (٢-١): أمثلة تطبيقية عن visual component

التطبيق	Category	البيــــــان
MemoPad	Standard	Memo pad text file editor. يظهر كيفية برمجة أوامر القائمة الرئيسية pop-upmenu وكيفية قراءة وكتابة الـ text files.
BitView	Standard، Additopnal، Dialogs	Bitmap file viewer. يفتح ويعرض أى Windows bitmap أو أى أيقونة أو ملف كبير أو أى ملف للرسومات آخر. ويعرض لك كيفية البرمجة باستخدام graphical and dialog component.
Dclock	Standard، System	ساعة رقمية. يوضح كيفية برمجة الـ timer component الذى يحدث فى نفس الوقت مع مهام أخرى.

تصميم التطبيقات باستخدام الـ component objects:

إبدء تنفيذ برنامج Delphi اذا لم يكن قد بدأت بعد، اذا كان تطبيقاً قد تم تحميله بالفعل، استخدم FilelNew Application لإضافة form جديدة للتطبيق، اضغط Standard page tab تحت الـ component palette لعرض ايقونات الـ component الموجودة فى هذه الفئة [راجع الشكل (٢-١)].

إختر ايقونة الـ component على سبيل المثال، اضغط الزر الذى يحمل حرف الـ A، والذى يمثل Label component حرك المؤشر داخل نافذة الـ form اضغط زر الفأرة الأيمن وأضف الـ Label على الـ form، فى الموقع الذى تريده ولا تقم بعملية الـ drag للـ component على form وبدلاً من ذلك أختر الـcomponent icon وحرك المؤشر الى الـ form وأضغط بزر الفأرة وأضف الـ object على الـ form أضغط مرتين متتاليتين على الفأرة أو على أى component

لإضافته على الـ form. بعد ذلك، اضغط واسحب الـ component لتحريكه.
لإضافة نسخ متعددة من نفس نوع component، إضف واحد على الـ form، وتاركاً إيها مختاره وأضغط Ctrl+C يتبعها Ctrl+V لكل نسخة جديدة.

لحذف الـcomponent، إختره واضغط Delete كن فى غاية الحذر وانت تفعل هذا - يمكنك ان تلغى آخر. اذا قمت بحذف الـ component، فإن الثانى فقط يمكن استعادته. لإسترجاع الـ object الذى قد تم حذفه إضغط Edit|Undelete(أو اضغط Ctrl+Z).

ملحوظة: لتفادى الغموض الذى قد يحدث فى هذا الكتاب، فإن مصطلح الـcomponent يعود على ايقونة فى VCL palette ; component object، (أو object فقط) هى عبارة component تم إضافته على الـ form. اذا صادفتك إحدى التعليمات فى هذا الكتاب تقول: اضغط object أو component object إختر من على الـ form كما وأن اسماء الـ Class تبدأ بحرف الـ T ؛ لذا فإن برمجة Label component توجد فى TLabel class. والـ Object قد تم تسميتها باستخدام اسماء الـ component وتعد Label1، Label2، FileLabel امثلة للاسماء الجيدة لانها تذكرك بأنها Label component object.

إضف component object قليلة على الـ form وذلك باختيارها من الـStandard palette واضغط الفأرة على الـ form بعد ان تضيف الـ object (لا يهم العدد بالضبط والنوع)، قم بتجربة التجارب التالية لتعتاد على اوامر ترتيب الـ component:

- لتحريك وإعادة تحديد حجم الـ component، أولاً اختره بالفأرة. (يقوم Delphi تلقائياً باختيار الـ component الى ثم إضافتها حديثاً) وهذا يحيط الـ component بمربعات صغيرة، تسمى handles. لتغيير حجم الـcomponent المختار، اضغط واسحب الـ handles لتحريك الـ object على الـ form اضغط على الـ component واسحب الفأرة.

- اذا تم إختيار الـcomponent (له visible handles) ولكن لاتستطيع استخدامه فعلى سبيل المثال لو ضغطت Delete تجد أنها لا تؤدى الوظيفة المطلوبة-

اضغط على الـ form لتنشيطه، ثم حاول ثانية. إننى احب ان استخدم مفتاح F12 لتنشيط الـ form.

● اضغط واسحب الفأرةعلى الـ form يتكون مستطيل حاول ان تجعل الـ component يقع داخل المستطيل وأترك الفأرة سوف تجد ان كل الـcomponent التى تقع داخل المستطيل قد أصبح لها handles. يمكنك عندئذ الضغط والسحب على أى object من objects التى لها handles لتحريكها كلها. يعرضDelphi dim handles حول كل component objects المختارة. لايمكنك استخدام هذه الـ handles لتغيير حجم الـobjects- يمكنك تغيير حجم الـcomponent object كل على حده.

● وهناك طريقة اخرى لاختيار الـ objects اضغط الأول لاختياره وأبق ضاغطاً مفتاح Shift وانت تضغط مؤشر الفأرة على الآخرين اختبر Edit|Select All اذا كنت تريد اختيار كل الـ component objects ووضعها على الـ form.

● ومع العديد من الـ objects المختارة، اضغط أى واحدة منها لتوقف اختياره. فى بعض الحالات، لكى تختار مجموعة كبيرة من الـ objects، من السهل ان تختارهم جميعاً ثم توقف اختيار ما لا تريده منها.

● لايقاف إختيار الـ objects، اضغط الفأرة على خلفية الـ form، أو مع أختيارك لـ forms أخرى أو اضغط Esc (هذه طريقة جيدة للاستخدام عندما يملأ الـcomponent objects أو أكثر الـ form، مما يصعب أو يستحيل معه ضغط خلفية النافذة). ويتم إيقاف إختيار جميع الـ component objects عندما لا ترى اية handles من المهم ان تعرف هذه الحالة لانه بامكانك ان تعدل الـ events والـpropertie الخاصة لـ form فقط عندما لايكون هناك component objects مختارة.

● لمحاذاة component objects متعددة بصورة مرتبة، اخترها واختر أمر الـ Edit|Align، الذى يعرض .dialog box. استخدم الازرار الموجودة فى هذا الـ dialog لمحاذاة الحواف ولضبط المسافات بطريقة مساوية فى المواضع المخصصة، ولإعادة ترتيبات أخرى. جرب الأختيارات الموجودة لتعتاد على اوامر المحاذاة.

● عندما يغطى احد الـ component objects على الآخر. استخدام Edit|Bring to Front و Edit|Send to Back لتغيير ترتيب العرض النسبى لل

objects . لتجربة هذه الأوامر ، اضف CheckBox objects قليلة على الـ form، ثم أضف GroupBox، اذا قمت بتحريك الـ GroupBox فوق الازرار، فإنه يغطيها . ولكى تجعل الازرار تظهر مرة أخرى، اختر الـ GroupBox واختر Edit|Send to Back لتنقله خلف الازرار . (قد تجد ان شيئاً لم يحدث، فقد يكون السبب هو انك اخترت الازرار وليس الـ GroupBox . لاختيار الـ GroupBox فقط ، أولاً قم بالغاء إختيار جميع الـobjects، ثم اضغط الـ GroupBox .

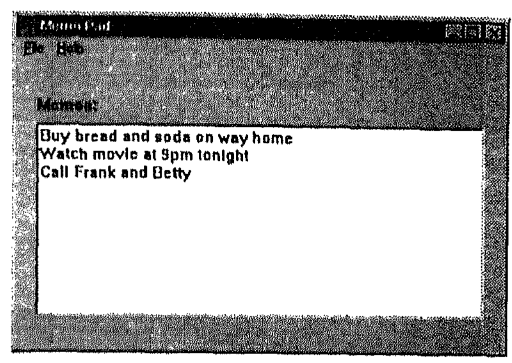
● اثناء التشغيل، يستطيع المستخدمين ضغط مفتاح Tab لتحويل الـ Focus من احد الـ controls الى الآخر . (يرجع الـ Focus إلىcontrol الحالى الذى يتلقى أى تعامل معه من لوحة المفاتيح) . لضبط التعامل مع tab order أختر بعض أو كل الـ component objects واختر أمر الـ Edit|Tab Order . وهذا يظهر dialog box وبإمكانك استخدامه لتحديد الترتيب الذى بموجبه يصبح كل control مستعداً لإستجابة الضغط على مفتاح Tab . ان أوامر الـ dialogs بديهية - ببساطة اضغط واسحب اسماء الـ component objects لإعادة ترتيبها أو اختر اسماً واضغط الزر الاسفل للـ dialog أو الزر الأعلى له .

عندما تختار اثنين أو أكثر من objects للـ component، تظهر نافذة الـ Object Inspector الخصائص والـ events المشتركة فيما بينهما . وأى تغيير فى هذه القيم تؤثر على كل الـ objects المختارة . على سبيل المثال لتغيير text style لمجموعة من check boxes أولاً، اخترها جميعاً ثم حدد أختيارتك لخاصية الـ form المشتركة .

Tip **فكرة:** هل تعرف انك تستعرض الـ events الخصائص المشتركة عندما تكون قائمة اللائحة الخاصة بالـ Object Inspector فارغه .

المثال التطبيقى MemoPad:

والمثال الخاص بهذه الفقرة، وهو الـ MemoPad، يوضح كيفية استخدام الـ Label و Memo ويظهر البرنامج ايضاً كيفية إنشاء menu bar للأوامر ويوضح شكل (٢-٢) ظهور الـ MemoPad .

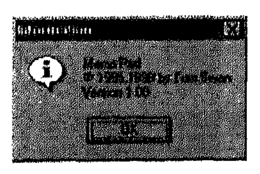


شكل (٢-٢): MemoPad يعرض الـ Label , Memo components

تشغيل الـ MemoPad:

لتشغيل تطبيق الـ MemoPad الآن، استخدم ....FilelOpen . أو اضغط زر Open-project لفتح الـ Memopad.dpr. اضغط F9 لتشغيل البرنامج. اكتب ملحوظات قليلة، ثم اترك الـ MemoPad وارجع الى Delphi حتى تستطيع فحص الـ components والـform الخاصة بالتطبيق.

اختر امر الـ FilelSave الخاص بالـ MemoPad لحفظ مذكرات فى text file يحمل اسم Memos.txt فى الدليل الحالى. ودائماً ما يستخدم الـ MemoPad اسم الملف هذا اختر FilelExit لإنهاءالـ MemoPad، والذى ايضاً يؤدى الى حفظ مذكراتك تلقائياً. (يمكنك ايضاً استخدام اية طريقة أخرى لترك البرنامج). ان اختيار HelplAbout يعرض display للرسالة الموضحة فى شكل (٢-٣)، والذى يعرف التطبيق.



شكل (٢-٣): dialog خاص بالرسالة الخاصة بالـ MemoPad تعرض عندما تختار أمر HelplAbout بالبرنامج

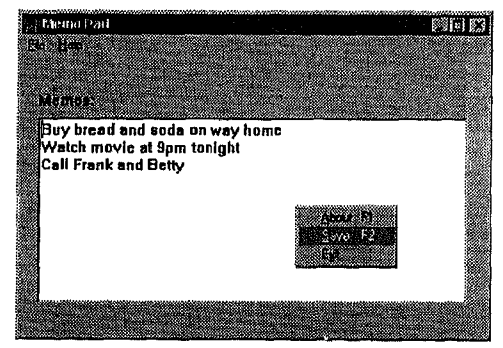
ان اثنين من فوائد الأوامر الخاصة بالمثال ولتلك الأوامر accelerator keys يمكن للمستخدمين ان يضغطوها لإختيار أوامر القائمة دون استخدام الفأرة. اضغط

F2 لحفظ مذكراتك . اضغط F1 لعرض display الخاص للرسالة . بالإضافة الى ذلك ، فإن اوامر القائمة تضع خطة تحت المفاتيح الهامة . على سبيل المثال ، اضغط Alt+F لاختيار .File|Exit .

ملحوظة: ان Delphi يقوم بإعطاء مصطلح خاطئ بعض الشئ لكل مفاتيح الـ shortcut فى لوحة المفاتيح . من الناحية التقنية ، يعتبر الـ accelerator هو مزيج من واحداً أو أكثر من الـ Ctrl ، أو الـ Alt أو مفاتيح الوظائف الأخرى المخصصة لأمر قائمة والموضحة الى اليمين منها فى menu's pop-up window و shortcut key هو حرف موضوع تحته خطاً فى قائمة أو امر القائمة .

وهذه ليست مشكلة كبيرة - فقط كن حذراً من ان المفاتيح التى تحتها خط الخاصة بأمر القائمة ، مثل الـ Alt+F X ، تختلف عن ccelerator keys مثل F1 أو Ctrl+X . وأى امر قائمة يمكن ان يكون له أى من النوعيين السابقين أو كلاهما .

ويستخدم برنامج الـ MemoPad ايضاً PopupMenu component لتوفير بديل آخر لتنفيذ الأوامر حرك مؤشر الفأرة فى أى مكان داخل نافذة الـ MemoPad- ولكن ليس على menu bar واضغط زر الفأرة الايمن . كما يوضح شكل (٢-٤) فإن هذا يفتح pop-up menu صغيرة .



شكل (٢-٤): pop-up menu الخاص بـ MemoPad تظهر عندما تضغط زر الفأرة الأيمن

تعمل كالقائمة الرئيسية ولكن تظهر عند موضع الفأرة . لإغلاق نافذة القائمة هذه ، اضغط زر الفأرة الايمن خارج pop-up menu أو اختر أمراً (أو اضغط Esc) .

Tip

فكرة: إعطى المستخدمين دائماً methods مختلفة كثيرة لتنفيذ الاوامر العامة. بهذه الطريقة، يمكن للمبتدئين اختيار أوامر من قوائم التطبيق (وهى بسيطة نسبياً للتعود على كيفية استخدامها)، ولكن الخبراء يمكنهم ان يحفظوا shortcut keys ويستخدموا pop-up menu لتنفيذ الأوامر بسرعة.

إنشاء الـ MemoPad:

بعد ان تعتاد على تطبيق الـ MemoPad، استخدم Delphi لفحص الـ components و الـ forms للبرنامج. اخرج من الـ MemoPad الآن لتعود الى Delphi. اختر كلاً من component objects للبرنامج وبالفأرة أضغط على الـobjects (الزر الأيسر)، واستعرض الـevents وخصائصها فى الـ Object Inspector.

Caution

تحذير: تعود على ان تضغط مرة واحدة على الـ component objects التى على الـ form ففى كثير من الـ components اذا ضغطت مرتين عليها بالفأرة، فإن Delphi يقوم بإنشاء procedure فارغ فى الـ source code للبرنامج للـ events الافتراضى، والذى هو عادةOnClick. اذا حدث هذا مصادفة، يمكنك اختيار Edit|Undo لإستعادة أو مجرد تجاهل الادخال. واذا قمت مصادفة بإنشاء event handler, لا تريده، فلا تقلق- مادمت لم تضف اية عبارات أو تعليقات إلى الـ procedure، فإن Delphi يمحوها تلقائياً فى المرة التالية التى تقوم فيها بتنفيذ المشروع.

استعرض ايضاً العبارات الموجودة فىMain unit (النافذة التى تحمل العنوان Main .pas). اذا لم تعثر على هذا النص، قم باظهار نافذة الـ code editor View|Units أو اضغط زر Toggle Form/Unit. أو، يمكنك ضغط F12 مرة واحدة أو عدة مرات حتى تظهر النافذة. يحتوىMain.pas على source code للـ Object Pascal، والتى يطابقها Delphi مع الـ form والـ visual component objects الخاصة بها. واثناء تطوير التطبيق، فسوف تنتقل غالباً بين الـ form و الـ source code التابع لها.

من المفيد ان تفكر فى الـ form و الـ unit التابع لها على انهما يقدمان شكلان مختلفان لنفس الـ objects، وتوضح الـ form المظهر الخارجى للـ controls

editor code تعـرض الـ Object Pascal Commande والتى تؤدى انشطة الـ components مثل الاستجابة الى ضغطة زر ما أو حفظ بيانات فى ملف. وكل form لديهـا unit واحـدة (والتى قـد تسـتخـدم units أخـرى) قـد تحـتوى على procedures و functions وتعريفات اخرى متعددة للـ component objects التابعة للـ form.

لا تهتم بفهم كل البرمجة الموجودة فى ملف الـ Main .pas الخاص بالـ MemoPad، والموضح هنا فى القائمة (٢-١). ان هذا الملف وغيره من الملفات الموجودة فى هذا الكتاب قد تم وضعها فى قوائم للرجوع إليها كمراجع فقط. ان Delphi قد أنشأ معظم البرمجة الموضحة هنا - لبرمجة مهام التطبيق، ولقد قمت بكتابة عبارات داخل الـ procedure declarations التى قام Delphi بإدخالها فى الـ component editor للتطبيق. وتعتبر هذه واحدة من خصائص Delphi الرئيسية للتطوير السريع للتطبيق - لإنشاء تطبيق، إختر component ثم إملأ حرفياً الفراغات التى يقدمها Delphi.

القائمة (٢-١): الـMemopad\Main.pas

```
unit Main;

interface

uses
  Windows, SysUtils, Messages, Classes, Graphics,
    Controls, Forms, Dialogs, Menus, StdCtrls;

type
  TMainForm = class(TForm)
      MainMenu1: TMainMenu;
        PopupMenu1: TPopupMenu;
        File1: TMenuItem;
```

```
        Save1: TMenuItem;
          Exit1: TMenuItem;
          Help1: TMenuItem;
          About1: TMenuItem;
          About2: TMenuItem;
          Save2: TMenuItem;
          Exit2: TMenuItem;
          MemoLabel: TLabel;
          Memo1: TMemo;
          procedure Save1Click(Sender: TObject);
          procedure Exit1Click(Sender: TObject);
          procedure About1Click(Sender: TObject);
          procedure FormActivate(Sender: TObject);
          procedure FormClose(Sender: TObject;
      var Action: TCloseAction);
            private
     { Private declarations }
         public
     { Public declarations }
         end;

   var
    MainForm: TMainForm;

   implementation

   {$R *.DFM}
```

```
procedure TMainForm.Save1Click(Sender: TObject);
begin
 Memo1.Lines.SaveToFile('memos.txt');
end;

procedure TMainForm.Exit1Click(Sender: TObject);
begin
 Close;
end;

procedure TMainForm.About1Click(Sender: TObject);
begin
 MessageDlg(
'Memo Pad'#13#10' ÷ 1995, 1998 by TomSwan'#13#10'Version 1.00',
  mtInformation, [mbOk], 0);
end;
procedure TMainForm.FormActivate(Sender: TObject);
begin
 if FileExists('memos.txt')
     then Memo1.Lines.LoadFromFile('memos.txt')
        else Memo1.Lines.SaveToFile('memos.txt');
end;

procedure TMainForm.FormClose(Sender: TObject;
 var Action: TCloseAction);
begin
 Memo1.Lines.SaveToFile('memos.txt');
end;
```

الآن، حـاول ان تعـيـد إنشـاء تطبيق الـ Memopad بنفسك. اذاتعـثـرت، استخـدم الملفات الموجـودة على القرص المدمج فى دليل الـ Source\MemoPad كمرشد لك، أو إرجع للقائمة (٢-١). إتبع هذه التعليمات خطوة بخطوة:

١- اسـتـخـدم الـ Windows Explorer لإنشـاء دليل مـثـل C:\Projects\Memopad لحفظ ملفات تطبيقك. اذا لم تكن تريد حفظ التطبيق، يمكنك تخزين الملفات فى دليل يحمل اسم Temp واحذفها فيما بعد.

٢- إبدأ مشروع جديد. إضغط داخل للـ form لتختارها، وقم بتغيير خاصية الـ Caption فى نافـذة الـ Object Inspector لتـصـبـح الـ Memopad. من الآن فصاعدا، عندما اذكر تغيير خاصية، إختر أولاً الـ form أو component، واذا لزم الأمـر، اضـغط الـ Properties tab فى نافـذة الـ Object Inspector. ثم اختـر الخاصية وغير قيمتها، الموضحة الى اليمين من اسم الخاصية.

٣- قم بتغيير خاصيةName للـ form لتصبح MainForm. وهى كلمة واحدة بلا مسافة بينها اذا كتبت رموز لا يسمح بها داخل خاصية Name، فإن Delphi يعـرض رسالة خطأ. يجب ان تبـدأ الاسـماء باحـرف أو هذا الرمـز (-)، ويمكن ان تحتوى على حروف أو ارقام أو هذا الرمـز (-) فقط. ان الـ Name الذى تقـوم بإدخـالة يتـمـاثل مع الاسم المبـرمج للـ object's ولذا، فـإن فى الـ source code يمكنك الإشـارة للـ Form object عن طريق البرمـجـة من خـلال الـ main .form

٤- إخـتـر File|Save All، أو اضـغـط زر Save All لإنشـاء ملفـات القرص الخاصة بالمشروع. يمكنك الانتظار الى وقت لاحق لحفظ المشروع، ولكن ان تفعل هذا الآن يؤدى الى إنشاء اسماء ملفات يستخدمها Delphi لإنشاء عناصر متنوعة فى نص البرنامج. إننى انصحك ان تحفظ البرنامج باسرع ما يمكن بعد ان تطلق إسماً على الـ form الرئيسية به.

٥- ان Delphi يقدم لك أثنان من الـ file dialogs، واحداً تلو الآخر. فى الـ dialog الأول، والذى يحمل العنوان Save Unit1 As، تحـول الى الدليل من الخطوة رقم (١)، واكـتـب Main فى مـربع الـ File Name. اضـغـط Enter أو اضـغط OK. فى الـ dialog الثـانى، الذى يحـمل العنوان Save Project1 As،

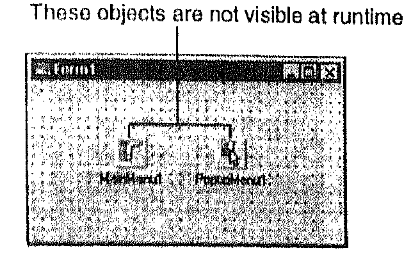
اكتب الـ MemoPad كأسم الملف . اضغط Enter أو اضغط OK . لقد اتممت إنشاء ملفين الآن : Main.pas و Memopad.dpr . يقوم Delphi بإلحاق امتدادات الـ pas. والـ dpr. لاسماء ملفاتك . ولإجراء الأختيار بشكل سريع ، يمكنك استخدام الاسماء الافتراضية حسب النظام وهى Project1 و Unit1 ، لكن فى أغلب الحالات ، سوف تريد تغييرها لتصبح اسماء ملف وصفية .

Tip

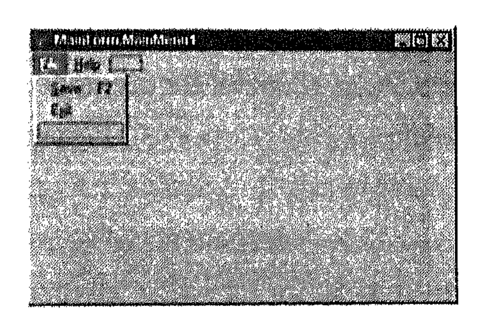
فكرة: يجب ان تكون اسماء ملف المشروع والـ Units تتبع القواعد الصحيحة فى الـ Pascal . ان اسماء الملفات الطويلة تكون مقبولة ، ولكنها قد لا يكون لديها مسافات بين الحروف أو رموز أخرى مثل علامات الترقيم .

٦- أضف MainMenu component ، على الـ form . أضف ايضاً PopupMenu component. كلاهما متواجدان فى Standard palette . فى هذا المثال ، استخدم أسماء الـ object المتاحة ، MainMenu1 و PopupMenu1 ، للتعرف على الـ objects . ولأنها components معقدة نسبياً ، فإن Delphi يعرضهما فى الـ form كايقونات تمثل مظهرهم النهائى [انظر الشكل (٢-٥)] . وتكون الايقونات غير مرئية اثناء وقت التشغيل . قم بتحريك كلتا الايقونتين الى الركن الايمن العلوى من الـ form ، أو الى أى مكان آخر ملائم . اذا لم ترى labels للأثنين كما هو موضح فى الشكل ، اختر Tools|Environment Options ، اضغط Preferences page tab ، وقم بإختيار الـ check box لعرض الـ component captions فى الـ Form Section designer . ويظهر الـ Captions فقط للـ components مثل الـ MainMenu و PopupMenu التى ليس لها labels أو شكل معين تظهر به . فالازرار ، على سبيل المثال ، والتى لها labels وهى من النوع الذى ترى أن له شكل مميز عن بقية الـ components وليس له أى Captions .

٧- لإنشاء اوامر القائمة ، اضغط مرتين MainMenu1 component object ، والذى يفتح الـ Menu Designer الخاص بـ Delphi ، كما هو موضح فى شكل (٦-٢) . لانشاء قائمة الـ File ، اكتب &File ، واضغط Enter لتغيير خاصية الـ Caption لشكل القائمة هذا . اكتب حرفاً مسبوقاً بعلامة (&) لتحديد الـ Alt shortcut key الخاص بالامر .



شكل (٢-٥): ايقوناتى الـ MainMenu و PopupMenu component يظهران فقط فى وقت التصميم - ولا يظهران فى نافذة البرنامج أثناء تشغيله. وتعليقات الايقونة الموضحة هنا تعتبر اختيارية



شكل (٢-٦): هذا هو الـ Menu Designer التابع لـ Delphi موضحاً الاوامر الموجودة فى الـ MainMenu object الخاص بالـ MemoPad

٨- يقوم الـ Menu Designer الآن بتحديد المكان التالى لإدخال أمر ، فى هذه الحالة ، تحت قائمة الـ File . إدخل &Save لإنشاء امر الـ Save . لا تضغط Enter (اذا فعلت هذا ، استخدم مفتاح السهم المشير الى أعلى لإعادة تحديد الأمر) . اختر خاصية الـ ShortCut فى نافذة الـ Object Inspector ، اضغط السهم المشير الى اسفل ، وأختر F2 accelerator shortcut key لهذا الأمر .

Tip **فكرة:** من الأسرع ان تكتب F و 2 وتضغط Enter بدلاً من ان تبحث عن مفاتيح تستخدم قائمة اللائحة الطويلة هذه التابعة لهذه الخاصية .

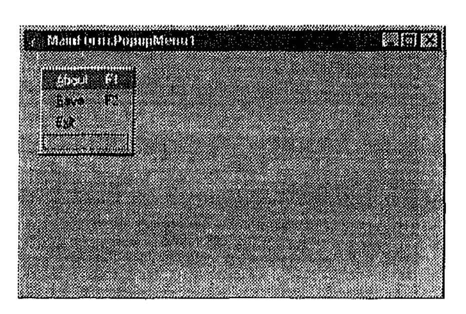
٩- اضغط المساحة الفارغة تحت أمر الـ Save فى الـ Menu Designer ، واختر خاصية الـ Caption فى نافذة الـ Object Inspector . ادخل E&xit لإنشاء مفتاح الـ Exit وتعيين الحرف Alt shortcut key X الخاص بالأمر .

فكرة: عند إدخال أوامر قائمة جديدة، اختر دائماً خاصية الـ Caption قبل ان تكتب نص الأمر.

١٠ - باستخدام اساليب مماثلة، ادخل قائمة الـ Help وأمر الـ About الخاص بها. ويكون F1 من accelerator shortcut key للأمر بنفس الطريقة التي عينت بها F2 للامر Save.

١١ - اغلق نافذة الـ Menu Designer لترى قائمتك في الـ form. يمكنك فحص هذه القائمة، ولكن في الوقت الحالي، لا تختر أمراً، مما يؤدي لإنشاء event handler في الـ unit module . (ولا يوجد ضرر اذا فعلت هذا من دون قصد).

١٢ - اضغط مرتين الـ PopupMenu component لتعيد فتح الـ Menu Designer، والذي يعرض هذه المرة نمط، ابسط للـ floating pop-up menus [انظر شكل (٢-٧)]. أضف أوامر الـ Exit والـ Save والـ About في هذه القائمة. ضع علامة & قبل الحروف التي تحتها خط. استخدم خاصية الـ ShortCut لكل أمر لتحديد accelerator shortcut keys مثل F1 و F2 .



شكل (٢-٧): الـ Menu Designer الخاص بـ Delphi موضحاً الاوامر الموجودة في الـ PopupMenu object الخاصة بالـ MemoPad

١٣ - لا يكفي مجرد إنشاء PopupMenu component يجب عليك أيضاً أن تخبر الـ form كيفية استخدامها اختر نافذة الـ Menu Designer، ثم اضغط الـ form لإيقاف اختيار أي component يجب أن ترى الـ MainForm: TMainForm في قائمة اللائحة الخاصة بالـ Object Inspector. قم بتغيير خاصية الـ PopupMenu التابعة للـ from لتصبح PopupMenu 1. ويمكنك كتابة هذا الاسم أو اختياره من

قـائمـة اللائحـة لهـذه الخـاصـيـة . وهـذا يجـعـل الـ from تعـرض PopupMenu 1 component عندما يضغط المستخدم زر الفأرة الأيمن على الـ from .

١٤- اضغط F9 لتشغيل التطبيق ، والذى لم يتم بعد حتى هذه المرحلة . إننى فى الغـالب أفعل هذا مباشـرة بعـد إنشاء أوامر برنامج حتى أتمكن من أخـتبار مظهرهم . حاول تجربة أوامر قائمة البرنامج (لا أحد منهم يعمل حتى هذه المرحلة)، واضـغط زر الفـأرة الأيمن لعـرض floating pop-up menu والتى لا تعـمـل هى الأخرى . للعودة الى Delphi ، اضغط Alt+F4 ، اضغط مرتين زر قـائمة النظام، أو اضغط زر إغلاق الـ Windows 95 .

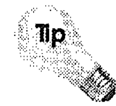
لقد أتممت الآن المرحلة الأولى من برمجة الـ MemoPad . بعد أن تستـرح قليلاً، قم بتحميل مشروع الـ MemoPad ، اذا لزم الأمر ، واتبع الخطوات التالية لإتمام التطبيق :

١- أضف الـ Label component على الـ form ، وقم بتـغـيـيـر الخـاصـيـة Name لتـصـبـح MemoLabel . اضـغط Enter ولاحظ أن Delphi يغـيـر الـ Caption تلقائياً تبعاً لتغير اسم الـ component .

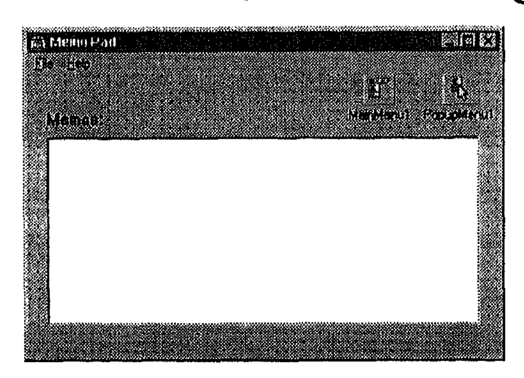
٢- فى الغـالب يكون من غيـر المستـحب للـ component أن يظهـر اسمـه الداخلى . لتغيير النص المعروض الخاص بالـ labels ، اختر الـ MemoLabel وغير خاصية الـ Caption لتصبح Memos :

٣- قم أيضاً بإضافة الـ Memo component على الـ form . فى هذه المرة ، يمكن أن تترك خاصية الـ Name تحدد حسب نظام البديل الإفتراضى للبرنامج ، وهو Memo1 . رغم ذلك ، قم بحذف كل الرموز من خاصيـة الـ Lines حتى تصبح مساحة الإدخال خالية . لكى تفعل هذا ، اختر خاصية الـ Lines للـ Memo1 ، واضـغط الزر البيضوى على اليمين . هذا يفتح محرر قـائمة String الخاصة بـ Delphi ، والتى يمكنك استخدامها لحذف السطر Memo1 (اضغط المساحة الخلفية عدة مرات متتالية أو قم تحديد هذا النص واضغط Del) اضغط Enter أو اختر زر OK الـ editor لتجعل التغيير ثابتاً .

٤- قم بإعادة تحديد حجم الـ Memo1 ورتب كل الـ components لتجعل العـرض الخـاص بك يتـلاءم مع شكل (٢-٨) . فى هـذه النقطة ، قـد تريد أن تحفظ المشروع وتشغله لترى المظهر النهائى للتطبيق .



فكرة: تستطيع أن تجرب تطبيقاتك. يمكنك أن تضغط F9 لتشغيل البرنامج فى أى مرحلة من مراحل تطويره.



شكل (٢-٨): استخدم هذه الصورة للـ MemoPad form فى Delphi كمرشد لك عندما تقوم بإنشاء تطبيق

٥- لبرمجة أوامر الـ MemoPad، اختر أمر الـ File|Save بالبرنامج فى الـ form. (لا تختر أمراً فى تطبيق أثناء تشغيله- اختر من خلال الـ form) عندما تفعل ذلك، يقوم Delphi بإضافة procedure بعيد عن الـ event handler من خلال الـsource code، وينتقل المؤشر تلقائياً الى editor. وهناك، يمكنك أن تكتب عبارة لأداء أعمال الأمر. ادخل العبارة التالية بين البداية والنهاية. ويجب أن يكون النص الواقع بين قوسين تحده علامات تنصيص فردية. لا تستخدم العلامة الزوجية. حول هذه البرمجة وغيرها من البرمجة المقترحة فى هذا الكتاب، إرجع الى ملفات أو قوائم القرص الخاصة بالمشروع [أنظر القائمة (٢-١) لهذا المثال].

```
Memo1.Lines.SaveToFile('memos.txt');
```

٦- وبالإنجليزية، فإن العبارة التى أدخلتها لتوك تحفظ سطور الـ Memo1 فى ملف يسمى Memos.txt. والـ Lines هو عبارة عن object يمتلكه Memo1 component object بمعنى، إن Lines هو نفسه object داخل الـ component والـ SaveToFile وهو عبارة عن method فى هذا الـ object يمكن لـ Lines أن يؤديه. والنقاط، وتسمى dot notation، تحدد أن Memo1، Lines، و SaveToFile مرتبطة ببعضها. ويمثل المقطع 'memos.txt' اسم ملف. والعبارة تمرر المقطع الموضح بين الأقواس الى الـ SaveToFile method. لاحظ أن العبارة

تنتهى بفصلة منقوطة . من الناحية الفنية ، وفى Pascal ، تفصل الفصلة المنقوطة إحدى العبارات عن الأخرى . فى هذه الحالة ، لأنه لا يوجد عبارة أخرى تالية ، يمكنك ترك الفصلة المنقوطة ؛ رغم ذلك ، فلا ضرر من أن تضعها . اذا وضعت الفصلة المنقوطة فى موضع خاطئ ، أو اذا لم تضع واحدة هامة ، فإن Delphi يخبرك بهذا فى رسالة خطأ عندما تقوم بتشيل البرنامج .

٧- اكمل اوامر الـ MemoPad الاخرى . اختر الـ File|Exit فى الـ form . إدخل ;Close بين كلمة البداية الرئيسية وكلمة النهاية الرئيسية (إرجع الى TMainForm.Exit1Click in Main.pas procedure) . كما عرفت فى الباب السابق ، ان الـ Close procedure ، واحد من الـ Delphi procedures التى لاتحصى والتى يمكنك إستدعاءها ، لتغلق النافذة . فى هذا المثال ، إذا كانت هذه النافذة الرئيسية للبرنامج ، فإن Close ايضاً ينهى البرنامج .

٨- إرجع الى الـ form ، واختر Help|About . ادخل العبارة التالية بين البداية والنهاية . اكتب بحذر كل حرف بالضبط كما هو موضح ، أو اذا أردت ان تغش ، انسخ والصق العبارة من ملف الـ Main.pas . (اننى أؤيد الغش دائماً ، فلا تترد فى ان تنسخ من هذه الصفحات أو من القرص المدمج المرفق . إنك لا تحتاج الى إذن خاص لتستخدم الـ source code (بهذا الكتاب فى عملك) . اكتب كل ما يلى فى سطر واحد - إنه مقسم على سطرين هنا بسبب حدود المسافة :

```
MessageDlg('Memo Pad'#13#10'@cw 1995,1998 by Tom
Swan'#13#10'Version 1.00', mtInformation, [mbOk], 0);
```

٩- ان العبارة تنشئ message dialog يعرض ملحوظة حق الطبع للـ MemoPad ورقم النسخة . وفى MessageDlg procedure ، يمرر البرنامج اربعة معاملات بين قوسين ، تفصلها فصلة . والمعامل الأول هو مقطع ، تجده علامتى تنصيص . فى هذا المقطع ، تقوم مجموعة العلامات #13#10 بادخال الـ carriage return و ASCII control codes لجعل النص الذى يكتب بعدها تكتب فى سطر جديد . لإدخال رمز حق الطبع ، استخدم Windows Character Map utility ، أو قص والصق الرمز من الملفات الموجودة على القرص المدمج . (أو ، مجرد ان تكتب C) . والمعامل الـ mtInformation للإختيار من بين عدة

انماط dialog styles ، والمعامل [mbOk] يعتبر مجموعة والتى لها فى هذا المثال عضو واحد، هو mbOk، يحدد نمط الـ dialog بواسطة زر OK . والمعامل 0 يعتبر شاغل المكان dialog's help context والذى لا يستخدمة الـ MemoPad . مرة أخرى، تنهى الفصلة المنقوطة العبارة . (لمزيد من المعلومات حول قيم معاملات MessageDlg، لمزيد من المعلومات عن الـ procedure إرجع الى Delphi's online help) .

١٠- لبرمجة اوامر pop-up menu، PopupMenu1 object (الذى عليه سهم) الذى قمت بإضفته على الـ form وهذا يعيد فتح نافذة الـ Menu Designer . أختر Events page tab فى نافذة الـ Object Inspector . اختر امر الـ About فى الـ Menu Designer، وحدد OnClick event بـ About1Click . يمكنك إدخال اسم الـ event handler هذا، اذا تم اختياره من قائمة لائحة الـ OnClick . حدد الـ OnClick event الخاص بأمر الـ Save بـ Save1Click . حدد الـ OnClick event الخاص بأمر الـ Exit بـ Exit1Click . إنها نفس برامج event handler التى قد برمجتها للقائمة الرئيسية- وهى إظهار لكيفية مشاركة أكثر من events فى نفس الـ code .

١١- اغلق نافذة الـ Menu Designer . اضغط داخل خلفية الـ form، واذا كان ضرورياً، اختر Events page tab فى نافذة الـ Object Inspector . اضغط مرتين بالفأرة المساحة الخالية الى اليمين من الـ OnActivate event، وادخل بعد ذلك عبارة الـ if-then-else الواقعة بين البداية والنهاية . وتعطى العبارة نص الملف . المعين فى component's Lines object :

```
if FileExists('memos.txt')
  then Memo1.Lines.LoadFromFile('memos.txt')
    else Memo1.Lines.SaveToFile('memos.txt');
```

والعبارة السابقة تستدعى الـ FileExists Function ، والتى ترجع True اذا لم يكن . اذا كان الملف موجوداً، فإن العبارة تعطى الملف فى الـ Lines الخاصة بالـ Memo1 . اذا لم يكن الملف موجوداً، فإن العبارة تحفظ Lines فى ملف جديد . ولأنه خاصية الـ Lines التابعة للـ object تعتبر فارغة فى هذه النقطة، فإن

استدعاء SaveToFile method الخاص به ينشئ ملف جديد خالى . بالرغم من أنه قد يبدو أن كثير من العمليات تحدث ، إلا أن بناء الـ if-then-else بأسره يعد عبارة واحدة ، فإن فصلة منقوطة واحدة تلزم لفصل هذه العبارة عن غيرها . اذا وضعت فصلة منقوطة فى نهاية السطر الثانى فإن Delphi يعرض رسالة خطأ عندما تقوم بتشغيل البرنامج .

١٢- اختر الـ form (من المحتمل أن يكون قد تم اختيارها) ، واضغط مرتين المسافة الواقعة الى اليمين من OnClose event . يقوم Delphi بإضافة procedure لهذا الـ event والذى يحدث عندما تغلق نافذة البرنامج . أدخل العبارة التالية بين البداية والنهاية لحفظ components Lines فى ملف مذكور :

```
Memo1.Lines.SaveToFile('memos.txt');
```

١٣- اضغط F9 لتشغيل البرنامج حاول تجربة الأوامر وارجع الى Delphi . قارن ملف الـ Main.pas للبرنامج الخاص بك بالملف الذى يحمل نفس الاسم على القرص المدمج الخاص بهذا الكتاب . قد تكون بعض الـ procedures فى ترتيب مختلف إعتماداً على الترتيب الذى أدخلت به الـ components فى form والترتيب الذى برمجت به الـ event handlers ، ولكن يجب أن تكون العبارات والشكل العام هو نفسه .

Dialogs Components و Additional Components:

فى هذه الفقرة ، تقوم بتطوير تطبيق لعرض الـ bitmaps الذى يستخدم الـvisual components من فئتين مختلفتين لـ Delphi ، وهما Additional ، و Dialogs ، اختر الـ page tabs المخصصة لهما واستخدم hint boxes لإيجاد مكان الـ mentioned المذكورة هنا . (تذكر : ضع مؤشر الفأرة على أيقونة component وانتظر ظهور hint boxes) .

من بين العناصر الأخرى ، تقدم فئة الـ Additional component Graphics التى يمكنك استخدامها لإنشاء bitmap و Graphics Device Interface (GDI) مثل المستطيلات والدوائر والخطوط . والـ method Windows GDI تعين الـ device-independent لرسم وطباعة الرسومات ويمكنك التوصل للـ GDI بإضافة component objects على الـ form .

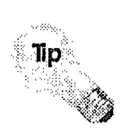
وتقدم الـ Dialogs فئة الـ components التي تكون الواجهة بين الـ Comman Windows و dialog Boxes . بالرغم من ان هذه هي كل Delphi component ، يقدم الـ Windows الـ code والمظهر الفعلي للـ dialog الناتجة . ان استخدام الـ dialog العامة يعطي للمستخدمين طرق مألوفة لتبادل دلائل الاقراص ، واختيار اسماء الملف ، اختيار الـ fonts والالوان ، والبحث عن أو أستبدال القيم في الوثائق .

استخدام الـ Image component :

لإنشاء الـ bitmap على الـ form أضف الـ Image component object من فئة الـ Additional جرب هذا الآن . اختر أولاً File|New Application لإخلاء مساحات العمل في Delphi ، ثم أضف الـ Image component object على form فارغة ولان الـ Image object ليس له run-time shape- ولأن محتوى bitmap يعرف مظهره- يعرض Delphi الـ Image component object شفافاً الذي لا يظهر في البرنامج النهائي وباختيار object outline Image، قم باختيار خاصية الـ Picture في نافذة الـ Object Inspector ، واضغط الزر البيضوي لفتح الـ Picture Editor الخاص بـ Delphi . أو ، يمكنك ان تضغط مرتين على الـ component object . استخدم الـ Picture Editor لإدخال أي bitmap داخل التطبيق . أولاً ، اضغط زر Load ثم اختر ملفاً ينتهي بالامتداد الـ bmp . (bitmap) ، ico. (icon) ، wmf . (Windows metafile) ، أو emf . (enhanced metafile) . يمكنك حينئذ ضغط الـ Save لنقل الملف على ملف قرص آخر ، أو الـ Clear لمسح الصورة التي تم تحميلها سابقاً ، أو الـ Cancel الالغاء آداء عمليات ، أو الـ OK لتحميل الصورة في Image component object .

اذا كنت مستمراً ، استخدم زر الـ Load للتعود الى دليل الـ Data ، على القرص المدمج . إفتح الايقونة Sample.ico ، واضغط OK . تظهر ايقونة الملف في Image component الخاص الـ form ، وتتغير خاصية الـ Picture لتصبح TIcon . أو ، لإدخال(bitmap) ، اختر ملف Sample.bmp يمكنك ايضاً تحميل صور أخرى اذا كان لديك بعض (bitmaps) مخزنة على القرص . قد يجب عليك

إعادة تحديد حجم Image object لترى (bitmap). ولتفعل هذا تلقائياً، حدد خاصية الـ Image AutoSize لـ True. يمكنك إعادة تحديد حجم bitmap's قم بتحميلها داخل Image component object، ولكن تبقى صورة الايقونة دائماً ثابتة الحجم.



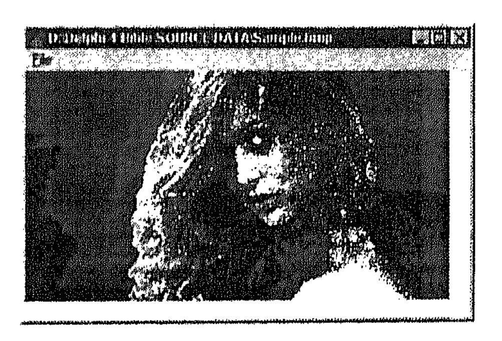
فكرة: بعد تحميل الـ bitmap داخل الـ Image object ، لتقليل أو توسيع الـ object ليلائم الـ bitmap بدقة حدد الـ AutoSize بـ True، اضغط مرتين Image object ، واضغط OK في الـ Image Editor.

التطبيق BitView:

ان التطبيق الموصوف في هذه الفقرة يجمع بين Standard والـ Additional والـ Dialogs لإنشاء برنامج لعرض bitmap.

تشغيل BitView :

لتجربة التطبيق، افتح ملف مشروع Bitview.dpr في الدليل الفرعي Bitview واضغط F9. اختر أي ملف ينتهي بـ bmp.. هناك مثال اذا احتجته في دليل الـ Data على القرص المدمج. شكل (٢-٩) يوضح الـ Bitview وهو يعمل.



شكل (٢-٩): تطبيق الـ Bitview هذا بامكانه ان يعرض أي bitmap file

سوف تقوم الآن بإنشاء تطبيق الـ Bitview الخاص بك. اذا قمت بتشغيل الـ Bitview، قم بانهاء البرنامج الآن قبل الاستمرار.

إنشاء الـ Bitview:

توضح القائمة (٢-٢) Main.pas source code لتطبيق الـ Bitview . استخدم هذا الملف كمرشد لك عندما تنشئ التطبيق باتباع التعليمات الموجودة بعد القائمة خطوة بخطوة .

القائمة (٢-٢): Bitview\Main.pas

```
unit Main;

interface

uses
  Windows, SysUtils, Messages, Classes, Graphics,
   Controls,
    Forms, Dialogs, Menus, ExtCtrls;

type
  TMainForm = class(TForm)
      BitImage: TImage;
        MainMenu1: TMainMenu;
        File1: TMenuItem;
        Open1: TMenuItem;
        Exit1: TMenuItem;
        OpenDialog1: TOpenDialog;
        procedure Open1Click(Sender: TObject);
        procedure Exit1Click(Sender: TObject);
      private
      { Private declarations }
      public
```

```
      { Public declarations }
       end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}

procedure TMainForm.Open1Click(Sender: TObject);
begin
 if OpenDialog1.Execute then
    begin
      BitImage.Picture.LoadFromFile(OpenDialog1.Filename);
        Caption := OpenDialog1.Filename;
      end;
end;

procedure TMainForm.Exit1Click(Sender: TObject);
begin
 Close;
end;

end.
```

لإنشاء تطبيق الـ Bitview الخاص بك، اختر أولاً الـ File|New Project لمسح أى مشروع تم تحميله حالياً سوف تعتاد على خطوات البداية بمرور الوقت، لذا لن اكررها فى المستقبل. كما أننى أختصر بعض الخطوات التالية، مثل كيفية حفظ

مشروع. اذا كنت اكملت التطبيق السابق بنجاح، فلن تقابلك مشكلة مع هذا التطبيق:

١- قم بتغيير خاصية الـ Caption للـ form لتصبح Bitmap Viewer. قم بتغيير خاصة الـ Name لتكون MainForm .

٢- قم بتغيير خاصية WindowState للـ form من wsNormal الى wsMaximized باختيار تلك القيمة من قائمة اللائحة بالخاصية. هذا يؤدى لعرض نافذة الـ BitView فى نمط الشاشة الكاملة. وتختار القيم الأخرى حالات نافذة أخرى - يعرض الـ wsMaximized النافذة كايقونة، ويعرض الـ wsNormal النافذة فى حجمها المحدد فى وضع يقرره Windows .

٣- اختر فئة الـ Additional، وأضف Image component على الـ form، غير خاصية الـ Name للـ object لتصبح BitImage. بالنسبه لموقع الـ object وحجمه لا يهم بدقة.

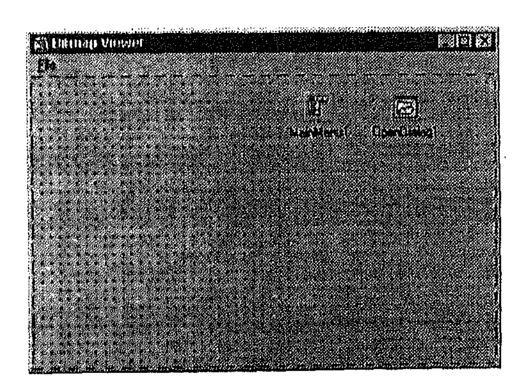
٤- ومع استمرار اختيار BitImage object، قم بتغيير خاصيته الـ Align لتصبح alClient. وهذا يؤدى الى إعادة تحديد حجم الـ component تلقائياً بنفس حجم الـ client area الخاصة بالـ form، وهى المسافة الواقعة داخل حدود النافذة.

٥- ولتجعل الـ bitmap الموجودة داخل الـ component object تتلائم ايضاً مع الـ client area وهذا يتطلب تغيير آخر. قم بتحديد خاصية الـ BitImage component's Stretch، اضغط مرتين قيمة الخاصية لتحويلها من الـ False الى True. (يمكنك ايضاً كتابة أو اختيار True أو False من قائمة الخاصية، ولكن الضغط مرتين على الخاصية True/False هو اسهل طريقة لتحديد قيمتها). ومع تحديد الـ Stretch بـ True، يعرض البرنامج الـ bitmaps فى حدود مساحة component. ولان الـ component object قد تم تحديد حجمه تلقائياً وفقاً للنافذة، فكذلك تكون الـ bitmap وهذا يظهر ايضاً كيف يكون الـ component object والبيانات الخاصة به، رغم علاقة كل منهما ببعضها، فى بعض الاحيان يتطالب البرمجة بشكل منفصل لكل منهما منفصلة.

٦- إضف الـ Standard MainMenu component فى الـ form والاسم الافتراضى حسب نظام البرنامج، وهو MainMenu1، واصفاً بما فيه الكفاية، ولا

يجب عليك تغييره اضغط مرتين ايقونة menu object ، وقم بانشاء قائمة تسمى File ذات أمرين، Open و Exit . تذكر ان تكتب علامة الـ & قبل الـ shortcut key، على سبيل المثال File&. اذا كان لديك مشكلة فى إنشاء قائمة البرنامج، راجع خطوات الـ MemoPad. لا يجب عليك تخصيص accelerator-shortcut keys ، ولكن تستطيع ان تفعل هذا اذا اردت. اغلق نافذة الـ Menu Designer. عندما ينتهى.

٧- من الـ Dialogs component palette، إضف OpenDialog component على الـ form. وكما يفعل Delphi للقوائم، فإنه يمثل الـ dialog كايقونات غير مرئية عندما يكون البرنامج فى حالة تشغيل. يمكنك وضع الـ MainMenu1 والـ OpenDialog1 فى أى مكان تريده. يجب ان يشبه ما يعرض على شاشتك شكل (٢-١٠)، والذى يوضح نافذة الـ BitView form بعد التطوير فى Delphi.



شكل (٢-١٠): استخدام هذه الصورة لنافذة الـ BitView form فى Delphi كمرشد لك عند تكوين الـ application

٨- إختر الـ OpenDialog1 component، واضغط الزر البيضوى المجاور لقيمة خاصية الـ Filter. هذا يفتح الـ Filter Editor الخاص بـ Delphi، والذى يمكنك استخدامه لتحديد الإمتدادات الخاصة بالملفات. على سبيل المثال، يمكنك تحديد الإمتداد bmp.* للفات، باستخدام الشكل (٢-١١) كمرشد. اضغط زر الـ OK بالـ dialog لحفظ الإمتدادات. (بعد هذه الخطوات المرفقة، سوف اصف طريقة أخرى للتعامل مع هذا الـ string).

شكل (٢-١١): Filter Editor Filename الخاص بـ Delphi لتبسيط إدخال الـ filename filters

٩- جميع الـ components الآن فى مكانها، وقد تريد حفظ، وتريد إجراء عمليتى الـ compile والـ link ومن ثم تشغيل البرنامج . اترك البرنامج لتعود لـ Delphi . (أوامر القائمة مازالت غير تامة، ولا تستطيع استخدام أمر الـ File|Exit لترك البرنامج، لذا اضغط زر إغلاق البرنامج بدلاً من ذلك).

١٠- فى نافذة الـ form- ليس فى البرنامج العامل- اختر File|Open لإنشاء procedure لهذا الأمر . وفيما بين كلمتى البداية والنهاية الرئيسيتين للبرنامج، أدخل البرمجة التالية (راجع الـ Bitview\Main.pas كمرشد، أو إفتح هذا الملف وانسخ عباراته) :

```
if OpenDialog1.Execute then
begin
  BitImage.Picture.LoadFromFile(OpenDialog1.Filename);
    Caption := OpenDialog1.Filename;
end;
```

١١- تقوم عبارة if السابقة بتنشيط الـ OpenDialog1 بإستدعاء object's Execute function . اذا كانت قيمة الـ function مساوية لـ True، فإن المستخدم قد أغلق dialog بإختيار زر الـ OK الخاص به . فى هذه الحالة، ينفذ البرنامج العبارتين داخل كلمتى البداية والنهاية الرئيسيتين . وتستدعى العبارة الأولى LoadFromFile method للـ Picture object لتحميل الملف المسمى من قبل خاصية Filename الخاصة للـ dialog . وتحدد العبارة الثانية نفس اسم الملف هذا

للـ form's caption، مما يغير عنوان النافذة الى مسار bitmap. ورمز الـ (=:) يعتبر عامل تخصيص من Pascal، والذى يخصص القيمة التى الى يمينه الى الـ object الذى الى يساره.

١٢- ارجع الى نافذة الـ form، اختر File|Exit، أدخل عبارة ;Close بين البداية والنهاية فى الـ event-handler procedure، الذى ينشئه Delphi للأمر.

١٣- احفظ المشروع بالكامل، واضغط F9 لإجراء عمليتى compile، والـ link لتشغيل التطبيق. قم بتحميل أى ملف bitmap. اضغط زر الحجم المتوسط على الحد الأيمن العلوى للـ BitView لتقلص النافذة الى حجمها الطبيعى. أعد تحديد حجم النافذة بواسطة فأرتك، ولاحظ أن bitmap تنكمش أو تتسع لتلائم حجم النافذة. هذا قد يجعل بعض الصور تبدو غريبة، [أنظر المشروع (٢-٤) فى نهاية هذا الباب لإقتراح تغيير هذه الخاصية.

لأن Image component قادراً على عرض أنواع متعددة من ملفات جرافيكية، لذا يسهل تحديث الـ BitView لعرض أيقونات، metafile وأنواع الصور الأخرى. ويمكنك أيضاً تحديد نوعية الـ standard file filter string ليمكنك التعامل مع display حيث يعرض bmp. و wmf. و emf. وملفات أخرى ذات امتدادات جرافيكية معروفة.

ولعمل هذا التغيير، ارجع الى Delphi واستخدم الـ Object Inspector لإختيار الـ MainForm. اضغط Events page tab، واضغط مرتين داخل المساحة الخالية الى اليمين من الـ OnActivate event وهذا ينشىء blank procedure for the event، والذى يحدث عندما تتعامل مع الـ form (فى بداية البرنامج، على سبيل المثال). أو الإنتقال من form الى الـ form. اجعل الـ procedure يبدو مثل هذا:

```
procedure TMainForm.FormActivate(Sender: TObject);
begin
  OpenDialog1.Filter := GraphicFilter(TGraphic);
end;
```

إن العبارة تحدد الـ GraphicFilter function التابعة للـ Graphics unit الى خاصية الـ OpenDialog1 Filter . وبدلاً من استخدام filter editor كما هو موضح من قبل ، فإن هذا يقدم كل امتدادات اسماء ملفات الجرافيك لـ TGraphic class .

وتغيير آخر يمكنك أن تقوم به هو أن تضيف صور JPEG الى الملفات القياسية . لفعل هذا ، حدد موضع uses declaration في بداية الـ source code ، وأضف JPEG unit الى units الأخرى الموجودة في قوائم . ويجب أن تبدو uses declaration مثل هذا :

```
uses
    Windows, SysUtils, Messages, Classes, Graphics, Controls,
      Forms, Dialogs, Menus, ExtCtrls, Jpeg;
```

والآن ، عندما تقوم بتشغيل البرنامج (اضغط F9) فإن الـ Open dialog يضع ملفات الـ Jpeg. و jpg. في قائمة مع أنواع ملفات الجرافيك المعروفة الأخرى .

:System Components

تقدم System component مجموعة متنوعة من الـ objects للتوصل الى system hardware and software . إن الـ Components الموجودة في هذه الفئة تتضمن Timer for background processing ، و PaintBox للجرافيك ذات الأغراض العامة ، و MediaPlayer component للوسائط المتعددة والـ components الأخرى تتضمن الـ OLE components والـ DDE لتشارك البيانات .

في النهاية ، أنك سوف تجد أمثلة لكل تلك الـ components في هذا الكتاب . في هذه الفقرة ، سوف تستخدم Timer لإنشاء ساعة رقمية تعد ٢٤ ساعة ، والتي تظهر أيضاً كيفية أداء background processes .

استخدام الـ Timer component:

إن الـ Timer component يعتبر واحداً من Delphi's simplest . فإن له خصائص قليلة و Single event يمكنك برمجة واحد أو أكثر لأداء مهام تتماشى مع تطبيقات أخرى ، أو تعمل في الخلفية .

لإنشـاء Timer component، اختـر من فئـة الـ System بـ component palette. أضف الـ component object على الـ form الحالية، قم بتغيير خاصية Name له اذا أردت ذلك، وقم بتحديد الـ Interval الذى ترغبـه لتكرار الـ event حسب المدة الزمنية وتقاس بالـ milliseconds وهى من 1 الى 65535. إن قيمة الـ 1000 تنشئ عـداد لكل ثانيـة؛ وقيـمـة الـ ١٠٠ تنشئ عـداد لكل ١٠/١ ثانيـة. (ويسمح لك Delphi بتحديد فترة العداد بصفر، ولكن لا يوجد سبب معقول لهذا).

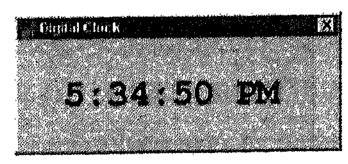
وللـ Timer component، event واحـد، وهو الـ OnTimer. اضـغط مرتين قيمة الـ event، أو يمكنك أن تضغط مرتين الـ Timer object، والذى يقوم بنفس الشئ، وأدخل عبـارات بين البـداية والنهـاية للتنفيـذ حسب Timer's frequency.

المثال: DClock:

لإظهار كيفية إنشاء Timer event handler، فإن هذه الفقرة تقدم مثالاً، وهو DClock. بالإضافة الى إظهار كيفية استخدام الـ Timer component، فإن هذا البرنامج التعليمى يوضح كيفية إنشاء نافذة ذات حجم ثابت، والتى تعتبر مفيدة عندما لا يكون هناك فائدة فى السماح للمستخدمين بضبط حجم النافذة.

تشغيل الـ DClock:

يوضح شكل (٢-١٢) عرض للـ DClock. وبخلاف الـ MemoPad، فإن نافذة الساعة ليس لها أزرار تكبير وتصغير، وليس لها قائمة أو أوامر.



شكل (٢-١٢): عرض الـ DClock يظهر background processing باستخدام Timer object

إنشاء DClock:

إفـتح ملف الـ DClock.dpr الآن وافحص form's component objects. افحص الـ Main.pas source code للبرنامج فى القائمة (٢-٣). بعد

الإنتهاء من الفحص، أغلق التطبيق وحاول إنشاء الساعة الرقمية الخاصة بك بإتباع التعليمات الموجودة بعد القائمة.

القائمة (٢-٣): Dclock\Main.pas

```
unit Main;

interface

uses
  Windows, SysUtils, Messages, Classes, Graphics, Controls,
    Forms, Dialogs, ExtCtrls, StdCtrls;

type
  TMainForm = class(TForm)
      TimeLabel: TLabel;
        Timer1: TTimer;
        procedure Timer1Timer(Sender: TObject);
      private
      { Private declarations }
      public
      { Public declarations }
      end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}
```

```
procedure TMainForm.Timer1Timer(Sender: TObject);
begin
  TimeLabel.Caption := TimeToStr(Time);
end;

end.
```

١- إبدأ مشروع جديد. قم بتغيير خاصية الـ Caption للـ form لتصبح Digital Clock وحدد الـ Name الخاص بها بـ MainForm. قم بإنشاء دليل مثل C:\Projects\Dclock لتخزين ملفات التطبيق، واحفظ المشروع. اجعل اسم ملف unit Main وملف المشروع Dclock.

٢- لإزالة أزرار التكبير والتصغير للنافذة، اختر الـ form (اضغط داخل نافذتها) واضغط مرتين خاصية أيقونات الـ BorderIcons. (ضع الفأرة على اسم الخاصية على اليسار، وليس على قيمتها على اليمين). لاحظ علامة الزائد الصغيرة الى اليسار من اسم الخاصية، والتى تتحول الى علامة ناقص بعد الضغط مرتين. وعلامة الزائد تخبرك أن هذه الخاصية لها واحد أو أكثر من القيم الفرعية المختبئة. وعلامة الناقص تشير الى أن القيم الفرعية تم عرضها. فى هذه الحالة، هناك ثلاث قيم من هذا النوع. حدد الـ biSystemMenu بـ True (القيمة الإفتراضية له حسب نظام البرنامج)، والقيم الفرعية المتبقية بـ False. بالرغم من هذا التغيير، مازالت الـ form فى Delphi توضح أزرار تكبير وتصغير. وهذا طبيعى لأنه يجب أن يمكن تغيير حجم النافذة عند تصميم التطبيق- يجب أن تقوم بتشغيل البرنامج لترى المظهر الحقيقى للنافذة، والذى هو فى هذه الحالة، محدد بحجمه المصمم.

٣- وتلك السمة تتطلب تغييراً آخراً- إزالة أزرار الحجم من النافذة وذلك لا يكفى لمنع المستخدمين من إعادة تحديد حجم إطار النافذة. لتثبيت حجم النافذة، اختر خاصية الـ BorderStyle الـ form، واختر bsSingle من قائمة اللائحة. لنفس السبب المذكور فى الخطوة رقم (٢)، فسوف ترى تأثير هذا التغيير فقط عندما تقوم بتشغيل البرنامج.

٤- أضف الـ Standard Label component على الـ form قم بتغيير خاصية الـ Name للـ object ليكون TimeLabel، وقم بتغيير الـ Caption الخاص به ليكون 00:00:00 AM وهذا يضمن أن الـ label لها الحجم المناسب لأرقام الوقت المدخلة فى وقت التشغيل.

٥- اختر خاصية الـ Font للـ TimeLabel. اضغط الزر البيضاوى لفتح حوار لإختيار الـ font. اختر الـ font، حجماً ونمطاً لساعتك. إننى قد خصصت الـ Bold Courier New مع حجم ٢٤ نقطة.

٦- أضف System Timer component على الـ form ولأن هذا هو Timer object الوحيد للبرنامج، فالـ Name الإفتراضى وهو، Timer1، يعتبر كاف. حدد الخاصية Interval لـ Timer object بـ ١٠٠٠، والذى يساوى ثانية واحدة، وهذه هى القيمة الإفتراضية.

٧- اختر Timer1 object، واضغط Events page tab فى الـ Object Inspector. يجب أن ترى event وحيد وهو الـ OnTimer. اضغط مرتين القيمة الموجودة الى اليمين من الـ event لإنشاء procedure فى source code. أو، يمكنك ببساطة أن تضغط مرتين الـ Timer object، الذى ينشئ برنامج الـ event handler- وهو الوحيد فى هذا الـ component. أدخل السطر التالى بين كلمتى البداية والنهاية الرئيسيتين :

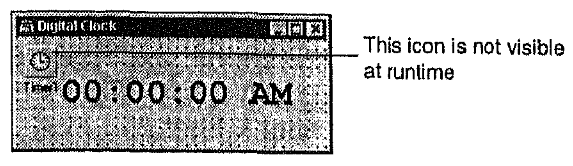
```
TimeLabel.Caption := TimeToStr(Time);
```

٨- والسطر السابق يحول الوقت الحالى الى string ويحدد النتيجة للـ Caption الخاص بالـ label وهذا يحدد الوقت على الشاشة مرة كل ثانية- وهو الزمن الذى تم تحديده للـ OnTimer event.

٩- قم بإعادة تحديد نافذة الـ object لتتلائم بشكل جيد. (استخدم الـ View|Form أو اضغط F12 لتجد النافذة اذا كانت مختفية). يجب أن يشبه شكل (٢-١٣)، الذى تظهر الـ DClock's form فى Delphi. وأيقونة Timer1 object لا تكون مرئية أثناء وقت التشغيل، ويمكنك تحريكها الى أى مكان ملائم.

١٠- اضغط F9 لإجراء عمليتى الـ compile والـ link وتشغيل التطبيق. فى ثوان قليلة يجب أن ترى ساعة رقمية تدق الوقت. اضغط Alt+F4 أو استخدم أمر

الـ System (أو اضـغط زر الإغـلاق فى الـ Windows 95) للخـروج والعـودة الى Delphi .



شكل (٢-١٢): استخدم هذه الصورة للـ DClock's form فى Delphi كمرشد لك عندما تنشئ تطبيق

:Win32 Components

أنك من المحـتـمل أن تسـتـخـدم component أو أكـثـر من Win32 component فى كل تطبيق تكتبه . وهذه الـ components تحـتوى على native controls ذى 32-bit الخـاص بتكوين أكـثـر من windows و dialog ، animations ، status bars ، coodbars .

وهذه الفـقـرة تقـدم إحـدى الـ Win32 components ، وهو TDateTimePicker ، والذى يمكنك إضـافته الى أى تطبـيق يحـتـاج الى طريقـة للمسـتـخـدمين لاختـيـار التـاريخ والوقت. وسـوف تـتـعـامل مع Win32 components الأخرى عبر هذا الكتاب .

والآن ، فإننا نركز على الـ TDateTimePicker إننى لا اعلم كم من الوقت أمضيت فى كتابة code للاستفسار من المستخدمين عن التاريخ والوقت ، ولكن مع TDateTimePicker component الخاص بـ Delphi ، لن احـتـاج الى إضـاعـة المزيد من الوقت فهـذا الـ component ليتح للمسـتـخـدمين ان يدخلوا ويختـاروا التـواريخ والمواقيت ولكنه يزيد عن مجرد المسـتفسـر - فهـو فى الواقع عبـارة عن component يحتوى على نظام تاريخى كامل .

والـ TDateTimePicker لا يمكن ان يكون ابسط استخداماً مما هو عليه . قم باسقـاط إحدى هذه الـ component على الـ form ، وحـدد خـاصـيـة الـ Kind التابعة له بـ dtkDate لتحديد اليوم والـ dtkTime لتحديد الوقت الحالى . لتعرف المزيد عن هذا الـ component ، إتبع الخطوات التالية :

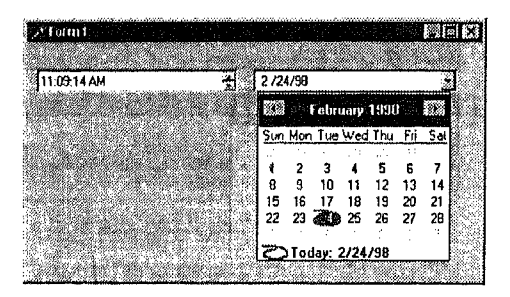
١ - ابدأ تطبيق جديد .

٢- اضغط Win32 page tab فى الـ Delphi's component palette .

٣- اختر الـ DateTimePicker component ، اضغط على الـ form ، لتضيف الـ object عليه ، إجعل خاصية الـ Kind مساوية لـ dtkTime .

٤- اعد الخطوة رقم (٣) لإضافة DateTimePicker آخر على الـ form . حدد خاصية Kind بهذا dtkDate (البديل الافتراضى) .

يمكنك تشغيل البرنامج فى هذه النقطة بضغط F9. يوضح الشكل (٢-١٤) عرض البرنامج و date component مفتوحاً (اختر قيمة للتاريخ بإختيارك لأحد القيم) .



شكل (٢-١٤): TDateTimePicker يبسط اختيار قيم التاريخ والوقت

بعد تجربة الـ component ، قم بإنهاء البرنامج الاختيارى وعد الى Delphi . قم بتجربة مجموعة من القيم الخاصة بالخصائص التالية :

- **Checked:** اجعل الـ check box الى اليسار من الـ control's edit وعند الاختيار، يتم التعامل مع الـ control ، وعند عدم الاختيار، يتوقف التعامل مع الـ control يستطيع المستخدمين إدخال أو اختيار التواريخ والمواقيت. وهذه الخاصية لا تمثل شيئاً الا اذا كانت خاصية الـ ShowCheckbox محددة بـ True .

- **Date:** التاريخ الذى يتم اختياره حالياً (والتاريخ حسب البديل الافتراضى هو هذا اليوم) مثل الـ TDate object استخدم هذه الخاصية لتحصل على تاريخ يتم اختياره من قبل المستخدم .

- **DateFormat:** اختر فيما بين Formats اما dfShort (٢/ ٢٤/ ٩٨) و dfLong (الثلاثاء ، ٢٤ فبراير ، ١٩٩٨) للعرض فى الـ control's edit وهذا القيم تؤثر فقط على الـ component ذات الـ Kind المحدد بـ dtkDate.

- **DateMode:** اختر بين نمطين لـ Date controls (Kind يساوى dtkDate) . استخدم dmComboBox (البديل الإفتراضى للنظام) ليستطيع ان يختار منها المستخدم تاريخا استخدم dmUpDown لعرض اسهم الإشارة الى أعلى واسفل والتى، عندما تضغط، تقدم أو تؤخر الجزء المختار من الوقت. بصراحة، إننى اجد ان هذا الخيار الثانى يصعب استخدامه، بالإضافة الى ذلك.

- **Kind:** كما ذكرت، حدد هذا بـ dtkDate لضبط التاريخ أو dtkTime لإدخال التوقيت. والخصائص مثل الـ DateFormat والتى تحتوى على لفظ الـDate فى اسمائها تكون ذات معنى للـ date controls فقط. رغم ذلك، حتى لضبط التوقيت، فإن مجال الـ Date لا يزال يقدم تاريخا صالحاً (وهو اليوم، الا اذا كنت قد غيرته من خلال نظام التشغيل).

- **MaxDate, MinDate:** أدخل الحدين الادنى والاقصى للتواريخ التى تريد ان يستطيع المستخدمين ادخالها فى control التاريخ (Kind تساوى dtkDate).

- **ShowCheckbox:** حدد هذه الخاصية بـ True لإظهار الـ check box الى اليسار من التاريخ أو التوقيت. وعند تشغيله (انظر خاصية الـ Checked)، فتصبح قادراً مع التعامل معه ويمكن تغييره. وعند إيقافه (يكون الـ checked مساوياً للـ false)، لا يستطيع المستخدمين تغيير قيمته. فكرة: استخدم خاصية الـ enable للتعامل مع الـ TDateTimePicker object أو إيقاف التعامل معه استخدم خاصيتى الـ Checked والـ ShowCheckbox اذا أردت أن تجعل المستخدمين قادرين على التعامل أو إيقاف التعامل مع الـ control.

- **Time:** الوقت الذى يتم اختياره حالياً (الآن هو الوقت حسب البديل الإفتراضى)، مثل TTime object. استخدم هذه الخاصية للحصول على وقت يتم إختياره من قبل المستخدم.

Tip

فكرة: قم بإبراز وحذف المدخل الإفتراضى حسب النظام فى خاصية Time. هذا يؤدى الى عرض 12:00:00 A.M. إن حذف خاصية الـ Date يحددها بـ ١٢/ ٣٠/ ٨٩. وهو السبت، ٣٠ ديسمبر، ١٩٨٩.

اذا كان برنامجك يحتاج الى تنبيه الى أن المستخدم قد إختار تاريخ جديد فقم بإنشاء الـ event handler للإستجابة عندما يتغير التاريخ المختار . هناك طريقتين لعمل هذا . اختر TDateTimePicker object، واضغط الـ Events page tab في الـ Object Inspector . اضغط مرتين الـ OnChange والـ OnCloseUp ، وأدخل الـ code لكلاهما (أو يمكنك ان تكتب procedure منفصل وتستدعيه من خلال الـ event handler) :

```
var
  S: String;
begin
  S := DateToStr(DateTimePicker1.Date);
     ShowMessage('Date entered = ' + S );
end;
```

ويقوم الـ procedure باستدعاء DateToStr (و function في SysUtils unit) لتحويل الـ Date الى string ، والذى يتم تمريره للـ ShowMessage لعرض التاريخ الذى تم اختياره فى نافذة dialog box من الواضح ضروره برمجة الـ OnChange والـ OnCloseUp حتى ان التغييرات اليدوية لقيم التاريخ وكذا والتواريخ المنتقاه من الـ object يتم تسجيلها .

افكار للمستخدم الخبير

• فى غالبية الحالات ، بعد إضافة الـ component على الـ form يجب ان تعطى للـ object إسماً يعكس هدفه . والإستثناء المعتاد لهذه القاعدة هو عندما تدخل نوع واحد من الـ component object ، على سبيل المثال ، اذا كان برنامجك له قائمة رئيسية واحدة ، فإن الاسم الافتراضى حسب النظام- MainMenu1 - يكون كافياً .

• استخدم اسماء واضحة مثل CloseButton ، و AgeLabel ، و NameEdit . بعض واضعى البرامج يودون استخدام اسماء قصيرة . على سبيل المثال B1 ، B2 ، B3 للازرار ولكن اذا فعلت هذا ، قد لا تتعرف على الـ objects

فى العام المقبل أو الشهر التالى - أو، اذا كانت ذاكرتك ليست افضل من ذاكرتى، غداً فى المساء.

● كقاعدة بديهية، ان برامجك سوف تصبح اسهل فهماً وحفظاً اذا الحقت اسماء الـ component بنهاية قيم خاصية الـ Name. على سبيل المثال، قد يكون اسم الـ NoteMemo افضل لـ Memo من اسم NoteMemo. فالاسم الاول يذكرك بنوع component object؛ والآخر يخفى هذه الحقيقة المهمة.

● يجب ان تبدأ الاسماء بحروف أو وضع خط تحت الكلمة، وقد تحتوى على حروف وارقام وخطوط تحت الكلام فقط. ويعتبر Label10 اسم صحيح، بينما 10Label ليس كذلك. لا يجب ان يكون فى الـ Names مسافات فيما بينها. استخدم حروف الكبيرة لتوضيح الاسماء الطويلة- مثلاً، يعتبر الـ MainSysMenu مقروءاً بشكل اوضح من الـ mainsysmenu. قد يحتوى الاسم على خطوط فيما بينه كما هو فى My_Main_Menu. ان الخطوط يصعب رؤيتها على الشاشة، رغم ذلك، ولا احبذ استخدامها. وقد تتكون الاسماء من واحد الى ٦٣ رمز ويجب ان تلتزم بتركيب الجمل المعرفة فى لغة Pascal.

● وبخلاف الـ Names، فإن الـ Captions وخصائص قيم النص الاخرى قد يكون لها أية رموز يمكنك كتابتها أو قصها أولصقها، فى أى ترتيب، بما فى ذلك المسافات وعلامات الترقيم. وطول الـ Caption محدد بـ ٢٥٥ رمز، ولكن من الناحية العملية، فإن حجم الـ objects يحدد طول الـ Caption الخاص به الى أقل من هذا بكثير.

● عندما تصبح الـ form ممتلئة بالـ component object، أو عندما يغطى object واحد أو اكثر مساحة الـ form، فقد تواجه مشكلة فى اختيار الـ form فى هذه الحالة، استخدم قائمة الـ Object Inspector's لاختيار الـ form بالاسم. على سبيل المثال، لاختيار الـ form الرئيسية فى معظم تطبيقات هذا الكتاب، اختر MainForm من قائمة الـ Object Inspector's. يمكنك استخدام هذا الاسلوب لاختيار الـ component object تم إضافته على الـ form.

● لمحاذة components متعددة، أولاً قم بتحريك component واحد الى وضعه الصحيح. اختر هذا الـ component أولاً ثم اختر الآخرين. ويقوم أمر



الـEdit|Align بمحاذاة الـcomponent object بالنسبة للـcomponent الذى اخترته اولاً .

● الـ Menu Designer ، اضغط Ctrl+Ins لإدخال أمر قائمة بين أمرين آخرين اضغط Ctrl+Del لحذف أمر . اكتب شرطة واحدة فى خاصية Caption لأمرها لإدخال فاصل بين امرين آخرين .

● اختر Edit|Lock Controls لتشغيل خيار هذه القادمة (تظهر علامة صح . بالقرب منه) . عند تشغيله ، تمنع هذا الخيار تحريك أو إعادة تحديد حجم الـcomponent object ، ولكن لا يزال بامكانك تعديل events وخصائص الـ control . بعد ان تقوم ببرمجةالـ form بالطريقة التى تريدها ، استخدم هذا الأمر لمنع إعادة الترتيب الذى قد يحدث للـ objects الموضوعة بعناية عن طريق الصدفة .

● اذا لم نتمكن من تحريك أو إعادة تحديد حجم الـcomponent object المختارة ، تنظر اذا ما كانت الـ Lock Controls المذكور فى الفكرة السابقة عاملة .

● والمستخدمين الجدد للـ Pascal دائماً ما يقلقون بشأن وضع الفصلة المنقوطة ، ولكن لا تبالغون فى إهتمامكم . اذا قمت بكتابة فصلة منقوطة فى مكان خطأ ، يقوم Delphi بكتابة رسالة Error فى الـ C والـ C++ ، تنهى الفصلة المنقوطة العبارات . فى Pascal ، تقوم الفصلة المنقوطة بفصل العبارات عن بعضها البعض . على سبيل المثال ، يمكنك الا تضع فصلة منقوطة قبل جزئية else فى عبارة if لأنه ، وعلى عكس الـ C والـ C++ ، الـ if-then-else الخاصة بـ Pascal تعتبر عبارة واحدة . إرجع الى قوائم هذا الكتاب للحصول على امثلة المواضع الفصلة المنقوطة الصحيحة . يمكنك ان تقضى بعض اللحظات فى الاختبار مع code editor's code insights والتى كما هو مفصل فى الباب الأول ، تقوم بادخال عبارات Pascal فى تركيب الجمل الصحيح تلقائياً .

● ولإضافة أكثر من الـ component objects من نفس النوع بطريقة اسرع ، اضغط مرتين الـ component icon فى الـ Delphi's palette . وهذا يؤدى لإضافة الـ object لهذا الـ component فى منتصف الـ form استخدم هذا الاسلوب لإضافة الـ component object ومتعددة – مجموعة من الـ check

boxes، مثلاً. وكل object يقع فوق الآخرين السابقين تماماً، ويجب ان تضغطهم وتسحبهم الى اماكنهم النهائية قبل ان تفعل شئ آخر.

● واسرع طريقة لإضافة الـ component object ومتعددة ومن نفس النوع هى ان تدخل واحداً، وتدعه مختاراً، ثم تضغط Ctrl+C يتبعها Ctrl+V لأى عدد من الـ objects ترغبه. وتعطى الـ object اسماء مرقمة متتالية مثل Label1، Label2، و Label3 .

● لإضافة component object على الـ form، اضغط مرتين على الـ object لإنشاء للـ event handler الخاص به. وهو دائماً أول event موجود فى نافذة الـ Object Inspector. بعض الـ component لا تعمل بهذا الشكل. فمثل، ان الضغط على الـ MainMenu مرتين نفتح نافذة الـ Menu Designer.

● لكى تجد الـ event handler متواجد، اضغطه مرتين الـ Events page بنافذة الـ Object Inspector. هذا يعرض الـ form's unit، ويضع المؤشر داخل event handler، ويقوم الـ Delphi بإنشاء event handler جديد فقط اذا لم يكن هناك واحداً موجود بالفعل.

● حدد خاصية الـ AutoSize للـ Label بـ True لتحصل على التحكم بتعديل حجمها تلقائياً لتلائم بيانات الـ Caption الخاصة بها. حدد هذه الخاصية بـ False اذا كنت تريد أن يظل الـ component ذا حجم ثابت فى كل الأوقات يجب أن يستخدم حجماً ثابتاً اذا حددت الـ WordWrap بـ True لعرض الـ labels متعددة السطور.

● إن طول الـ captions لنافذة أو طول single-line string تجد من الصعوبة عرضها لأن حجم الـ Object Inspector صغير ولعرض بصورة أسهل، أدخل النص فى أى مكان من نافذة code editor قم بإبراز النص، اضغط Ctrl+X لقصه الى الـ Windows clipboard، إرجع الى الـ Object Inspector، قم بإبراز خاصية نص مثل الـ Caption، اضغط Ctrl+V لإدخال نصك. إننى أود الإحتفاظ بمثل هذه الـ Strings العديدة كتعليقات فى ملفات الـ source code الخاصة بى، فيما بين قوسى تعليقات (* and *). يمكننى، عندئذ، عرض النص بسهولة ونسخه (اضغط Ctrl+C بدلاً من Ctrl+X) من الـ clipboard الى الـ

Caption . على سبيل المثال، أدخل تعليقات مثل التالى فى أى مكان فى unit's source code .

```
(*
This is the first window caption
This is another window caption
*)
```

● اذا كنت تريد أن تقوم بنفسك بتحليل مداخل الوقت والتاريخ فى TDateTimePicker لتقييد المداخل بتواريخ معينة، مثلاً، الأيام من الاثنين الى الجمعة- فقم بتحديد خاصية الـ ParseInput بـ True . وهذا ينفذ الـ OnUserInput event عندما يكتب المستخدم فى نافذة الـ edit .

المشروعات التى يمكنك تجربتها

(٢-١) : قم بتغيير خصائص متنوعة فى الـ component object لـ MemoPad . على سبيل المثال، عين Font مختلف لـ Memo1 object . قم بتغيير الـ MemoLabel، وعليك تجربة خاصية الـ Color التابعة له .

(٢-٢) : عندما تقوم ببدء تطبيق الـ DClock، فإنه يعرض أولاً 00:00:00 AM قبل إظهار الوقت الحالى . لتحسين العرض وإظهار الوقت بمجرد ظهور نافذة البرنامج، قم بإنشاء الـ OnCreate الخاص بالـ form . اضغط خلفية الـ form لإختيار الـ form، ثم اختر page Events tab فى الـ Object Inspector . اضغط السهم المشير لأسفل الى اليمين من الـ OnCreate، واختر Timer1Timer . هذا هو نفس الـ event handler الذى قمت ببرمجته لـ Timer1 . عندما تقوم بتشغيل البرنامج، فإنه يبدأ الـ TimeLabel بإستدعاء الـ label بالوقت الحالى قبل أن تصبح النافذة مرئية .

(٢-٣) : افتح مشروع الـ BitView، واختر الـ OpenDialog1 الخاص به . إن خاصية الـ Ctrl3D متوفرة للملاحظة الخلفية فقط، وفى الـ

Windows 95 و NT، لم يعد لها أى تأثير. إن جميع الـ controls فى هذه الانظمة العاملة لها خاصية 3D رغم ذلك، اذا كنت لا تزال تستخدم نسخة اولية من Delphi أو Windows 3.1 ، نحاول تغيير خاصية الـ Ctrl3D لـ False، واضغط F9. عندما تختار امر البرنامج FileIOpen، فإنك ترى Windows dialog box قياس، وغالباً ذا خلفية بيضاء ومظهر واضح نسبياً. للعودة الى تاثير الـ 3D، قم بتغيير خاصية الـ Ctrl3D مرة اخرى الى True.

(٢-٤): افتح مشروع الـ BitView واختر الـ BitImage. حدد خاصية الـ Stretch للصورة بـ False، واضغط F9 افتح ملف bitmap إنه يعرض بشكل اسرع لان تعديل احجام الصور يستغرق وقتاً. رغم ذلك، لسوء الحظ، ان عرض صور ذات الحجم الكامل قد يقتطع المساحات التى تمتد بعد حدود النافذة. فى الباب القادم، سوف تدخل scroll bars فى النافذة لحل هذه المشكلة. ايضا راجع الباب العاشر لمزيد من المعلومات حول scrolling .

الملخص:

• ان Visual component تبسط عملية برمجة الـ Windows. لإنشاء تطبيق Delphi، قم بإضافة الـ component object على الـ form، واختر الـ event وخصائص الـ objects.

• اختر فيما بين فئات الـ Delphi's component باختيار واحداً من page tab الموجودة فوق الـ component palette. اضغط component icon الذى تريده، قم بتحريك مؤشر الفأرة داخل الـ form، واضغط مرة اخرى لإضافة الـ component فى هذا الموضع.

• ان اختيار component يؤدى الى إحاطته بمربعات سوداء والتى يمكنك استخدامها لاعادة تحديد حجم الـ object. يمكنك ايضاً ضغط وسحب الـ component لوضعها على الـ form اختر component بضغطه مرة واحدة، أو اختره بالاسم من قائمة اللائحة فى الـ Object Inspector. يمكنك ايضاً اختيار components متعددة بضغط وسحب الـ outline الموجود حولها.

● ان الخصائص هى قيم تحدد سمات الـ components على سبيل المثال ، Buttons و Labels لها خاصية Font لاختيار اسم الـ form ، وحجم النقطة والنمط .

● تمثل الـ event افعال اثناء وقت التشغيل خاصة بالـ components . ان ضغط قيمة الـ event مرتين يؤدى الى إضافة الـ event-handler فى الـ source code لبرمجة الـ event إضف unit أو أكثر من عبارات Pascal بين كلمتى البداية والنهاية الرئيسيتن والتان موضوعتان تلقائياً بواسطة Delphi .

● عندما تختار أكثر من components ، تعرض نافذة الـ Object Inspector الـ event المشتركة والخصائص المشتركة فيما بينهما . وتؤثر أية تغييرات تفعلها فى هذه القيم على كل الـ components المختارة .

● يحافظ الـ Delphi على التمثيل البصرى لتطبيقك فى الـ form وتمثيل الـ source code لـ Object Pascal فى نافذةالـ unit ، كلاً التمثلين يتطابقان بحيث ان التغييرات فى إحدى النافذتين تنعكس فى الاخرى .

● تشير علامة زائد صغيرة الى اليسار من الخاصية الى ان هذه الخاصية لها قيم فرعية . اضغط مرتين حقل قيم الخاصية لوضع هذه القيم فى قائمة .

● ويشير الزر البيضاوى الى اليمين من قيمة الخاصية الى ان ضغط القيمة مرتين ، أو ضغط الزر مرة واحدة يفتح dialog boxes .

لتغيير موضع هذه الخاصية وبعض الخواص لها كلاً من القيم الفرعية و dialog boxes .

● استخدم الـ TDateTimePicker component من لوحة الـ Win32 لإنشاء control يستطيع من خلاله المستخدمون ادخال أو اختيار التواريخ والوقت .

فى الباب التالى ، سوف تعرف المزيد عن الـ form والخصائص و الـ event . سوف تعلم ايضاً كيفية الاستجابة لاعمال الفأرة كيفية إضافة الـ scroll bars للـ form وكيفية ان تجعل النافذة مستقرة على القمة ، وكيفية إنشاء splash screen .

*****





الباب الثالث

معلومات أولية حول الـ Forms

محتويات الباب:

- **Forms as components**
- **Form templates**
- **استخدام dialog النافذة الرئيسية**
- **إغلاق نافذة**
- **Form frameworks**
- **Data modules**
- **The splitter component**

إن كلمة الـ form، قد تصف أشياء كثيرة. فى Delphi كل مشروع له على الأقل form واحدة تمثل النافذة الرئيسية للبرنامج. كما يمكن أن يكون للمشروعات أيضاً form متعددة للنوافذ الصغيرة، dialog box، وشاشات لإدخال البيانات. بالرغم من أن الهدف الرئيسى للـ form هو حمل components أخرى مثل الأزرار، check boxes، ويمكن للـ form أيضاً أن يؤدى أعمال استجابة للـ event، مثل الضغط على لوحة المفاتيح والفأرة.

والـ forms تعتبر ضمن versatile objects الأكثر تنوعاً واستخدامها بكثرة لتطوير التطبيقات. فى هذا الباب، سوف تعرف الخصائص الرئيسية للـ forms وخواصها والـ events.

:Forms as Components

تمثل الـ form المظهر البصرى لنافذة، ولكن الـ form ليس مجرد نافذة أخرى جميلة. إن الـ form فى الحقيقة عبارة عن Delphi component لها مجموعتها الخاصة من الـ events والخصائص. وبخلاف الـ events الأخرى، رغم ذلك، لا تظهر الـ form فى component palette.

من الطبيعى أنك تنشئ الـ form بواحدة من إحدى الطريقتين: ببدء تطبيق جديد أو بإختيار أمر الـ File|New Form. استخدم المنهج الأول لإنشاء الـ forms الرئيسية لتطبيقك. واستخدم المنهج الثانى للحصول على forms إضافية- لإنشاء About dialog، مثلاً، أو لعرض شاشة بدء تشغيل واسعة. (المزيد عن هذه الموضوعات سيأتى لاحقاً).

:Forms and units

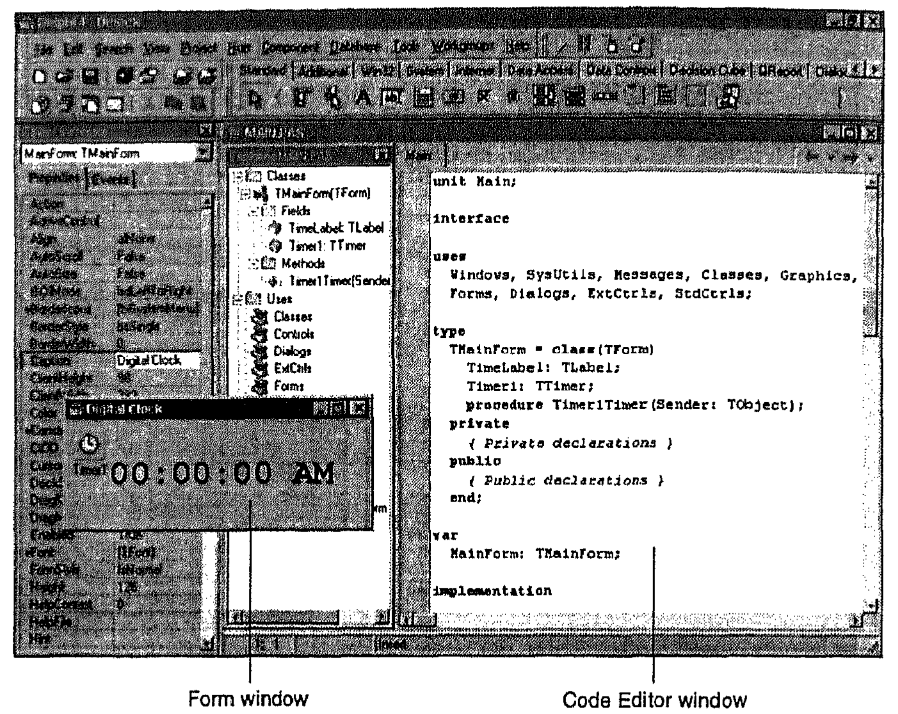
عندما تقوم بإنشاء forms جديدة، يضيف Delphi نافذتين لبيئة البرمجة. تظهر نافذة الـ form التمثيل البصرى للـ form كما تظهر فى التطبيق التام. وتضع نافذة الـ code editor قائمة ببرمجة الـ Object Pascal للـ form. والنص الموجود فى هذه النافذة يسمى unit. يوضح شكل (٣-١) نافذة الـ code editor ونافذة الـ form لتطبيق الـ DClock من الباب الثانى. وداخلياً، الـ form عبارة عن object unit، والتى تسمى فى بعض الأحيان module.

يقوم Delphi تلقائياً بإنشاء Object Pascal unit لكل forms جديدة تقوم بإدخال ما فى التطبيق. ليس من الممكن أو المستحب إضافة أكثر من form فى unit واحدة. يجب أن يكون لكل unit، form تقابلها. تقدم بعض الـ unit بيانات و code لـ modules أخرى لتستخدمها.

Caution

تحذير: الى واضعى البرامج الخبراء فى Pascal، كونوا على حذر! إن محاولة أكثر من form فى unit يمكن أن يعوق البرمجة البصرية والإمكانات لإنتاج الـ code التابعة لـ Delphi. للحصول على أفضل النتائج، إلتزم بقاعدة form واحدة لـ unit واحدة. ان Delphi لا يضيف أكثر من form فى الـ unit الواحدة، فإن الطريقة الوحيدة التى تستطيع أكتساب الخبرة

من هذا الخطأ هي بكتابة units جديدة من لاشئ، أو بمحاولة جمع أثنان من الـ form units في واحدة. فلا تفعل هذا.



شكل (٣-١): Delphi يحتفظ بنافذة الـ form، تحمل labeled Digital Clock والبرمجة المرتبطة بها في نافذة الـ code editor (التي تحمل عنوان Main.pas)

حفظ الـ forms في مشروعات:

عند حفظ مشروع جديد، لا تقم بتحديد خاصيةName للـ form وإسم ملف الـ unit بنفس الاسم. على سبيل المثال، اذا حددت خاصية الـ Name لـ form بـ MainForm، لا تقم بحفظ الـ form unit على أنها Mainform.pas (إنني اقترح استخدام Main.pas كبديل). لان الأسم الداخلي للـ unit يساوى إسم الملف الخاص بها ناقص الامتداد pas.، فإن استخدام نفس الاسم للـ form وال unit ينتج عنه تضارب في الاسم.

لتجربة هذا الخطأ حتى تتلافاه في المستقبل، إتبع هذه الخطوات :

١ - ابدأ تطبيق جديد وقم بتغيير خاصية الـName للـ form لتصبح Main.

٢- احفظ المشروع، وادخل Main كأسم ملف الـ unit. يقوم Delphi تلقائياً باضافة الامتداد pas. لإنشاء اسم الملف الفعلى، Main.pas.

٣- إضغط زر OK الخاص بـ file-save dialog أو اضغط Enter. بسبب تضارب الاسماء، فبدلاً من حفظ الملف، يعرض Delphi رسالة خطأ، "البرنامج يحتوى بالفعل على form أو module تسمى Main". إضغط OK لإغلاق نافذة رسالة الخطأ.

٤- تحل المشكلة بتغيير خاصية الـ Name للـ form لاسم يختلف عن اسم الملف الخاص بها- على سبيل المثال، MainForm. يمكنك الآن ان تعيد الخطوة رقم (٢) وان تحفظ الوحدة كـ Main.pas.

لمنع هذا النوع من تضارب الاسماء، والذى قد يتكرر حتى تعتاد على قواعد التسمية فى Delphi، إتبع الأفكار التالية لاختيار خصائص الـName واسماء ملفات الـ unit:

- أضف كلمة Form لكل Form;s name property والأمثلة على اسماء الـform الجيدة هى MainForm للنافذة الرئيسية للبرنامج، AboutForm للـ About dialog، وEntryForm لشاشة إدخال البيانات.
- إحفظ الـ unit modules مستخدماً الـName التابعة لهم ناقص كلمة Form على سبيل المثال، عندما تقوم بحفظ المشروع، حدد اسم الملف .Main للـMainForm module، و About للـ AboutForm، وEntny للـEntryForm . وهذا يؤدى الى إنشاء الملفات الثلاثة Main.pas وAbout.pas وEntry.pas .

خصائص الـ form المختارة:

تعرض بعض النماذج من خلال هذا الكتاب كثير من الخصائص للـ form object. الكثير من الخصائص لها وظائف معروفة، ولذا لن اشرحها مرة أخرى هنا. (فخصائص الـ Height والـ Width للـ form لا تحتاج لشرح وظائفها). وكذلك إنك قد قابلت كثير من خصائص الـ form مثل الـ Caption والـ Name والـBorderStyle. وفيما يلى اقوم بشرح خصائص الـ form-object الأكثر غموضاً والتى يمكنك رؤيتها وتحريرها فى نافذة الـ Object Inspector .

Note

ملحوظة: كن على حذر من ان بعض الخصائص قد ورثتها الـTForm class من class اخرى مثل TComponent وبشكل عام، هذا ينطبق على غالبية component class اذا بحثت عن class فى وثائقك المطبوعة، فإنك تحتاج ان تفحص التسلسل الأعلى لتجد كل الخصائص والـ methods للـ class .

● **ActiveControll:** حدد هذه لأى component object مثل الـ Button أو CheckBox، باعطاء الـ object الـ input Focus عند بدء ظهور الـ form، ان عملية الـ input Focus ليس بالشئ الثابت، فيمكن للمستخدمون أن يضغطوا Tab لتحريك الـ input Focus من component لآخر .

● **AutoScroll:** حدد بـ True لتظهر وتختفى الـ scroll bars تلقائياً فى نافذة حتى لو ان هناك مزيداً من المعلومات للتعامل مع window borders ان هذه هى القيمة الافتراضية حسب النظام، والتى تؤثر ايضاً على الـ Form Scaling اذا كانت الـ AutoScroll محددة بـ False، فإن كلاً من مساحة الـ client area ومستطيل النافذة قد تم تغيير مقياسهم، واذا كانت True، فإن الـ client area للـ form فقط هى التى يتم تغيير قياسها وغالباً مع الـ Scaled form التى يتم تغيير قياسها، يجب عليك ان تحدد الـ AutoScroll بـ False . لمزيد من المعلومات حول استخدام الـ scroll bars انظر الـ "Form Frameworks" فى هذا الباب .

● **Cursor:** يعين شكل المؤشر للعرض عندما يتحرك مؤشر الفأرة فى الـ client area للنافذة اختر إحدى القيم من القائمة لهذه الخاصية .

● **Enabled:** عادة تكون محددة بـ True لكى تستجيب الـ form للفأرة أو الـ timer، أو الـ Key Board event حددها بـ False لإيقاف الـ event handling لا تفعل هذا للـ form. حدد هذه الخاصية بـ False فقط للـ form التى يتحكم فيها البرنامج . (ان اسلوب splash-screen الموضع فى هذا الباب يستخدم هذه الخاصية) .

● **HorzScrollBar:** حدد القيم للـ scroll bar الأفقى فيظهر الـ scroll bar اذا كانت القيمة الفرعية Visible قيمتها بـ True وكان الـ Range اكبر من الـ

ClientWidth. اضـغط مرتين اسم الخاصية (ليس قيمتها) لفتح القيم الفرعية هذه.

- **Icon**: تختار ايقونة لتمثل الـ form اذا كانت الـ form هي النافذة الرئيسية للبرنامج، تحدد خاصية الـ Icon أيقونة النظام التى يعرضها الـ Windows عندما تقوم بتصغير النافذة.

- **KeyPreview**: تحـدد بـTrue اذا كنت تريد ان تتلقى الـ form اغلب الـ keyboard event قبل الإنتقال إلى أى Focused control اذا كانت False، فإن كل الـ keyboard event تذهب للـ control المختـار بغض النظر عند تحديد الـ KeyPreview، ان الـ form لا تتلقى ابداً الـ keyboard event كـالتى تحـدث نتيجة الضغط على Tab، والـ BackTab ، ومفاتيح الأسهم، الا اذا كـان الـ control الحالى (مثل input box، مثلاً) يتعامل مع هذه المفاتيح.

- **Menu**: على وجه الخصوص الـ MainMenu component يستخدم كـ menu bar للـ Delphi form بشكل طبيعى وتستخدم هذه الخاصية للـMainMenu الاول التابع لـ form، ولكن يمكنك تغييره لقائمة اخرى. انظر الباب الخامس لمزيد من المعلومات.

- **ObjectMenuItem**: يستخدم مع تطبيقات الـ OLE. لمزيد من المعلومات انظر الباب السادس عشر.

- **PixelsPerInch**: تحدد كيف يقوم التطبيق بإنشاء الـ form ليتم قياسها بالنقاط لكل بوصة. استخدمه مع Scaled لإنشاء form لتبدو متشابهة فى الحجم فى Screen Resolution مختلفة. اذا كـان الـ Scaled محـدد بـ False، فإن الـPixelsPerInch لا يكون لها أى تأثير. واسم هذه الخاصية الى حد ما مضلل لان قيمته لا تساوى عدد النقاط فى البوصة الواحدة. وكبديل، فإن Delphi يحدد الـ PixelsPerInch على أساس حجم font النظام، ثم يستخدم هذه القيمة لقياس الـ form.

- **Position**: تحدد الـ method لحساب حجم ومكان الـ form والتحديد الأفتراضى حسب النظام، وهو poDefault، يعرض النافذة فى حجمها وموضعها



المصمم، فإن الـ poDefaultPosOnly يحدد المكان للنافذة كما هو مصمم، ولكن يحسب حجمها فى وقت التشغيل، (يحدد الـ Windows الحجم المبدئى لـ form فى وقت التشغيل)؛ أما الـ poDefaultSizeOnly تحدد حجم النافذة كما هو مصمم؛ ولكن الموقع يحدد اثناء وقت التشغيل (يحدد الـ Windows اين تظهر النافذة)؛ والـ poScreenCenter يجعل النافذة فى الوسط اثناء العرض.

• **Scaled**: تحدد بـ True لاستخدام خاصية الـ PixelsPerInch وانشاء الـ form التى يتم تغيير مقاساتها تلقائياً بالنسبة لحجم الـ form هذا يساعدك على إنشاء تطبيقات تبدو جيدة بغض النظر عن موقع العرض. تحدد بـ False لعدم تغيير مقاسات الـ form هام جداً لضمان ان النص يعرض بشكل سليم فى Form's control على انظمة تستخدم fonts كبيرة. وهذا يعتبر هام للغاية فى الـ Windows 98 والـNT 5.0، والتى يستطيع المستخدمون اختيار قياسات الـ form عشوائياً.

• **Tag**: لا يوجد تحديد مسبق. استخدم الـ Tag لتخزين أى قيمة عدد صحيح تريده- مثلاً، رقم نسخة، أو code أو عدد يفحصه البرنامج اثناء وقت التشغيل.

• **VertScrollBar**: تماما مثل الـHorzScrollBar، ولكن يشكل الخصائص الفرعية للـ scroll bar الرأسى للـ form.

• **Visible**: تحدد بـ True ليجعل الـcomponent مرئياً؟ أو False ليخفيه حتى يستدعى البرنامج Show method للـ component object. خاصية الـ Visibl للـ False form عادة تكون لان التطبيقات تجعل الـ form الخاصة بها مرئية تلقائياً فى الوقت المناسب وللـ form المستخدمة نافذة صغيرة dialog box، يمكنك تحديد الـ Visible بـ False لإخفاء النافذة لحين الحاجة.

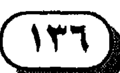
• **WindowMenu**: فى تطبيقات الـ Multiple Document Interface (MDI)، تصمم القائمة لعرض عناوين للنوافذ المفتوحة. حدد هذه الخاصية لأى بند قائمة (عادة تلك التى تحمل عنوان "Window") فى شكل الـ MainMenu للـ form.

ملحوظة: انظر الباب الخامس عشر، تطوير تطبيقات الـ MDI، لمزيد من المعلومات حول تطبيقات للـ MDI واستخدام الـ form لإنشاء الـ child windows.

الـ form events المختارة:

كما هو الحال فى خصائص الـ form، فإن النماذج الموجودة فى هذا الكتاب تظهـر الكثيـر من للـ events، والبعض الآخـر- مثـل OnClick والـ OnDblClick- تكون واضحة فيما يلى الـ event الأقل وضوحاً المعروضة فى نافذة الـ Object Inspector للـ form المختارة:

- **OnActivate:** يتم استدعائه عندما يقوم البرنامج بالتعامل مع أى form، وعندما تتلقى لأول مرة input focus مثلاً عندما تنتقل الى البرنامج من تطبيق آخر. انظر ايضاً الـ OnDeactivate..
- **OnClose:** يتم استدعاءه عندما تغلق الـ form، انظر ايضاً الـ OnDestroy..
- **OnCloseQuery:** يستدعى قبل ان تغلق الـ form مباشرة، غالباً عندما يستدعى التطبيق الـ Close procedure. يمكنك استخدام هذا الـ event لمنع فقد البيانات سؤال المستخدمون عن موافقتهم على حفظ الملفات التى تم تغييرها قبل ان ينتهى التطبيق، أو لمنع form من ان تغلق. لمزيد من المعلومات، انظر "إغلاق نافذة" فى هذا الباب.
- **OnCreate:** يستدعى مرة واحدة عندما يقوم البرنامج بإنشاء الـ form object فى الذاكرة. استخدم هذا الـ event لتنفيذ البدء مرة واحدة.
- **OnDeactivate:** يستدعى عندما يتحول المستخدم عن التطبيق. انظر الـ OnActivate. ايضاً.
- **OnDestroy:** يتم استدعاءه قبل تدمير الـ form object استخدم هذا الـ event، لإنهاء تواجد أى resource يمتلكها النظام اذا كانت الـ form هى نافذة البرنامج الرئيسية، فإن الـ OnDestroy يعتبر فرصتك الأخيرة لأداء أى عمل قبل إنتهاء البرنامج.



• **OnDonner, OnBlitzen**: تحدث عند حدوث أى event مهما كانت للـ form .

• **ODockDrop, OnDockOver**: يحدث الـ event إستجابة لحدوث أسلوب docking أما بالنسبة للـ events الذى لهم علاقة بهما . مثل الـOnEndDock، والـ OnStartDock ، والـ OnUndock . لمزيد من المعلومات، انظر "إنشاء الـ Docking Controls" فى الباب الثانى عشر .

• **OnDragDrop, OnDragOver**: جزء من أسلوب drag-and-drop الخاص بـ Delphi . لمزيد من المعلومات عن انتقال البيانات، والـ DDE والـ OLE ، انظر الباب السادس عشر، أسلوب التعامل مع Clipboard ، والـ DDE والـ OLE .

• **OnHide**: يستخدم لأداء اعمال عندما تكون الـ form مختفية. فمثلاً، يستطيع الـ OnHide محو الذاكرة أو أىresources أخرى عندما تكون الـ form فى حالةnot visible .

• **OnKeyDown**: يستدعى عندما يضغط المستخدم أى مفتاح، بما فى ذلك مفاتيح التى لها وظيفة المفاتيح الخاصة. استخدم هذا الـ event فى حالة التعامل مع أكثر مع مفاتيح فى نفس الوقت، مثل ضغط مفاتيح الـ Alt والـShift ، مع مفتاح أخرى يمكن للبرنامج ان يتلقى أكثر من OnKeyDown events دون حدوث الـ OnKeyUp events فيما بين ذلك. قم دائماً بتحديد الـ KeyPreview بـ True عند استخدام هذا الـ event .

• **OnKeyPress**: تحدث عندما يضغط المستخدم single ASCII أو الـcontrol key ولكن ليس مفتاح له وظيفة أو أى مفتاح خاص. استخدم هذا الـ event. فى حالة الضغط على حرف من لوحة المفاتيح وهذا الـ .event يكون بعد الـ OnKeyDown ولكن قبل الـ OnKeyUp . قم دائماً بتحديد الـ KeyPreview بـ True عند استخدام هذا الـ .event.

• **OnKeyUp** : يستخدم عندما يترك المستخدم أى مفتاح، بما فى ذلك المفاتيح التى لها وظيفة والمفاتيح الخاصة. الـ OnKeyUp event يتبع حدوث الـ

OnKeyDown استـخـدم هذا الـ .event بمفـرده مع الـ OnKeyDown لتنظيم نشاط لوحة المفاتيح - على سبيل المثال، فى لعبة تؤدى اعمال (من المحتمل من خلال الـ Timer) وبينما يستمر المستخدم فى ضغط مجموعة مفاتيح معينة. ويحـدث الـ OnKeyUp لكل OnKeyDown ، ولكن ليس من الضـرورى ان يكون هذا فى نظام صـارم مـتـعـاقب تحت - فـوق - تحت - فـوق (يرسل الـ Windows دائماً رسائل مفاتيح تحت ومفتاح فوق بالترتيب ولكن ازواج الرسائل قـد لا تكون متعـاقبة - تحت١ ، تحت ٢ ، فـوق ١ ، فـوق ٢ قـد يحـدث) قم دائماً بتحديد الـ KeyPreview بـ True عند استخدام هذا الـ .event .

- **OnMouseDown**: يحـدث عندما يضغط المستخدم أى زر الفـأرة ويتم وضع مـؤشـر الفـأرة فـوق الـ client area للـ form ويسـتـدعى الـ Delphi الـWindows API GetCapture للـ OnMouseDown لضـمـان ان الـ object يتلقى الـ OnMouseDown event و OnMouseUp المتلاحقة . وهذه السمة يتم التحكم فـيـهـا بواسطة الـ csCaptureMouse فى خـاصـيـة الـ ControlState لـ TControl class.

- **OnMouseMove**: يستدعى عندما يحرك المستخدم مؤشر الفأرة داخل client area للـ form . يمكنك ايضاً تحديد ما اذا كان المستخدم قد ضغط مفتاح الـ Shift، أو Alt ، أو Ctrl اثناء تحريك الفأرة.

- **OnMouseUp**: يستـدعى عندما يترك المستخدم زر الفأرة الذى قد انتج سابقاً الـ OnMouseDown event ، بغض النظر عن وضع مؤشر الفأرة.

- **OnPaint**: يتم استـدعاءه عندما تحتاج محتويات الـ form الى عملية الـ updating- على سبيل المثال، بعد ان يقوم المستخدم بتحريك نافذة اخرى جانباً. لمزيد من المعلومات عن برمجة الـ Graphics ، انظر الباب الثالث عشر، "تطوير تطبيقات الـ Graphics" .

- **OnResize**: يستدعى بعد ان يتغيير حجم النافذة وقد يكون هذا استجابة لسحب زر حجم النافذة، وتصغير أو تكبير النافذة. لاحظ ان النافذة يتم إعادة تحديد حجمها بالفعل عندما يتلقى تطبيقك هذا الـ .event .



• **OnShow**: يستدعى قبل ان تصبح النافذة مرئية . استخدم هذا الـ event. لتنفيذ ما تريده عند ظهور النافذة .

للتعامل مع مفتاح الـ Tab من خلال الـ OnKeyDown، راجع باستمرار الـ Shift flags . اذا ضغط المستخدم Alt+Tab للأنتقال من تطبيق الى آخر، يتلقى الـ OnKeyDown مجموعة المفاتيح هذه. اذا لم تتأكد من الـ Shift flags والذى يحتوى على Alt فإن برنامجك قد يتداخل مع نظام التشغيل الـ Windows .

Form Templates:

يمكنك إقامة الـ form الخاصة بك من لا شئ، أو يمكنك الاختيار من واحدة من عدة form templates، سابقة البرمجة يتم تقديمها مع Delphi. وهذ templates يسهل استخدامها وتعديلها، وهى متميزة جداً خاصة فى إنشاء انماط نموجية وفى التشغيل .

كما سيتضح فى الفقرات التالية، يمكنك ان تنشئ form templates خاصة بك لتوفر قاعدة مشتركة لبرامج جديدة مثل شاشات إدخال البيانات custom dialog boxes، وتطبيقات أخرى .

استخدام الـ form templates :

قم بتجربة الخطوات التالية لإنشاء الـ About dialog باستخدام الـ form templates سابقة البرمجة تعرض معلومات حول حق الطبع ومعلومات اخرى عن تطبيقك .

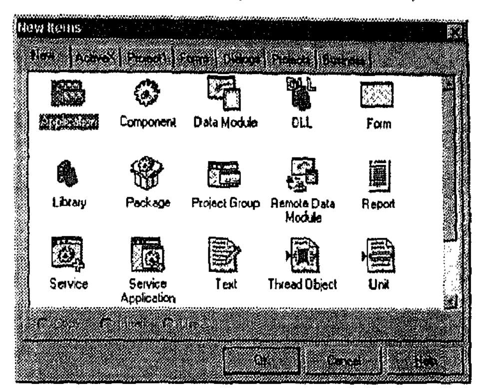
Note

ملاحظة: ان النسخ الأولى من Delphi تتطلب اختيار الـ Options|Environment وتشغيل خيار الـ Gallery . استخدم الـ New Form، لإحضار الـ dialog من الـ form templates المتاحة والنسخ الاحدث من Delphi تعرض تلقائياً Dialog مشابهاً، رغم انه اطول، عندما تختار أمر الـ .....New.

١ - ابدأ مشروع جديد، وقم بإنشاء دليل مثل C:\Projects\Aboutex للملفاته . قم بتغيير الـ Name للـ MainForm form والـ Caption الخاص به ليكون About Box Example . احفظ المشروع فى الدليل الذى انشأته إجعل اسم الـ unit Main والمشروع بـ Aboutex .

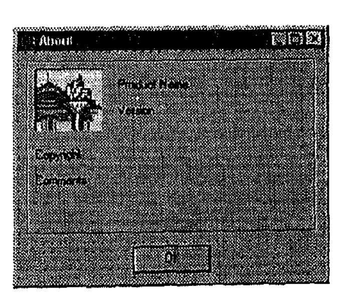
٢- لإضـافة form ثانيـة للمشـروع، اختـر FilelNew....As الموضح فى شكل (٣-٢)، وهذا يعـرض الـ New Items dialog (والذى يحـمل عنوان Browse Gallery) والذى يضع قـائمـة بالـ form و templates اخـرى. يوضح جـدول (٣-١) الـ templates المختـارة فى New Items dialog- سـوف ترى البعض الآخر خلال هذا الكتاب.

٣- اضـغط الـ Forms page tab فى نافـذة New Items dialog. اخـتـر الـAbout Box template بالفـأرة أو بضـغط لـلـ Tab أو مفـاتيح الاسـهم. اضـغط الـEnter أو اضغط OK لإضافة الـ form فى التطبيق واغلق الـ New Items dialog.



شكل (٣-٢): New Items dialog form و templates أخرى

٤- يجب ان ترى نافذة About dialog الخالية الموضحة فى شكل (٣-٣). حسب النظام الأفتراضى، تعتبر خاصية الـ Name للـ form محددة بـ AboutBox. قم بتغيير هذا الـ Name ليكون AboutForm. وقم بتغيير ايضاً الـ Caption الخاص بالـ AboutForm ليكون About This Program، أو الى أى نص آخر.

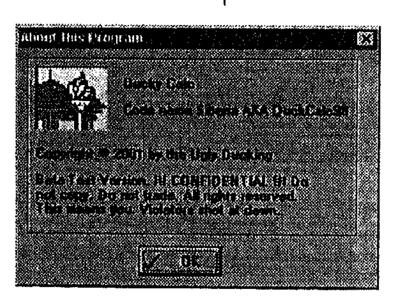


شكل (٣-٣): نافذة AboutBox form-templates الغير معدلة

Note **ملحوظة:** اننى اقترح إحتواء اسم الـ component ، كما يحدث فى الـ form بالنسبة للخاصية Name التابعة له .

٥- احفظ المشروع ، وعندما يعرض الـ save dialog Delphi ، إضف الـ About لإسم ملف AboutForm unit . هذا ينشيء الـ Object Pascal module جديد ، وهو About.pas ، فى الدليل الحالى . ان للبرنامج الآن أثنان من الـform واحدة للنافذة الرئيسية للبرنامج(MainForm) وواحدة للـ About dialog (AboutForm) . وكل form لها Object Pascal module تابعة لها : Main.pas للـ MainForm و About.pas للـAboutForm . بالإضافة الى ذلك ، فإن الملفMain.dfm يخزن خصائص الـ MainForm; والـ About.dfm يخزن خصائص الـ AboutForm . قد تريد ان ترى إسماء ملفات المشروع مع Windows Explorer فى هذه المرحلة .

٦- قم بتعديل الـ Label component objects الخاصة بالـ AboutForm كما تريد . إدخل اسم برنامجك ، وملحوظة حق الطبع ، وأى نص آخر تريده . يمكنك ايضاً إضافة Label إضافى و components اخرى فى نافذة الـ AboutForm . وأية تغييرات تقوم بها تؤثر على هذا التطبيق فقط - فلا تتأثر الـform template ، ويظل صالحاً للاستخدام فى المستقبل . يوضح الشكل (٣-٤) مثال من الـ AboutForm dialog التام .



شكل (٣-٤): الـ AboutForm التام من تطبيق الـ AboutEx

٧- اذا قمت بتشغيل البرنامج فى هذه المرحلة ، فإنه يعرض نافذة فارغة فقط . رغم ان الـ MainForm يصبح مرئياً بطريقة تلقائية ، فإن الـ AboutForm dialog يختفى لحين الحاجة إليه . لعرض dialog ، اضف أولاً للـ Button

component للـ MainForm . احتفظ بالـ Name الافتراضى، وهو Button1، وقم بتغيير الـ Caption ليكون "Click Me!,"، أو أى نص آخر. حدد حجم الزر ونافذة الـ MainForm كما ترغب. إضغط مرتين الـ Button1 object فى الـ form لإنشاء الـ OnClick event handler، واضف هذه العبارة بين البداية والنهاية:

```
AboutForm.ShowModal;
```

٨- وتقوم العبارة السابقة بإستدعاء ShowModal function التابعة للـ AboutForm. وهذا يعرض modal dialog، وهو الذى يجب عليك إغلاقه قبل الاستمرار فى استخدام التطبيق، و modeless dialog يسمح للقوائم والاوامر الأخرى بأن يعمل بشكل طبيعى بينما يظل الـ dialog مفتوحاً. يوضح الباب عشر الكثير عن الـ modal والـ modeless dialogs.

٩- اذا حاولت إجراء عملية الـ compile لتطبيق، فسوف تتلقى رسالة الخطأ لان الـ Main unit لم تتعرف بعد على الـ AboutForm object فى About module. جرب هذا- ان النسخ الحديثة من Delphi يمكنها أن تضيف البرمجة اللازمة لاستخدام الـ AboutForm unit. أجب بنعم Yes اذا سؤلت. أو، لإدخال البرمجة يدوياً، اضغط الـ Cancel بنافذة الخطأ واظهر Main unit فى code editor (يمكنك الضغط على Ctrl+F12 واختر Main من View Unit dialog. حدد موضع أمر الاستخدامات فى أعلى الـ Main، وهو يضع قائمة بالـ modules الأخرى التى تستخدمها هذه unit وبين اسم الـ modules الأخيرة والفصلة المنقوطة، ضع فصلة واكتب كلمة About. تأكد من أن الأمر مازال ينتهى بفصلة منقوطة على سبيل المثال، إن الـ uses directive الخاص بك قد يبدو مثل ما يلى، ولكن وعدد الـ modules بالطبع قد يختلف إعتماداً على التعبيرات التى قمت بها فى نافذة الـ AboutForm. والمدخل الجديد موضح هنا بخط سميك:

```
uses
  Windows, Messages, SysUtils, Classes, Graphics,
    Controls, Forms, Dialogs, StdCtrls, About;
```

١٠ - والآن، بما أن الـ Main يدخل، أو يستخدم About unit، فإن العبارة السابقة التى تستدعى الـ ShowModal. اضغط F9 لإجراء عملية الـ compile، والـ link لتنفيذ التطبيق. عندما تظهر نافذته الرئيسية، اضغط الزر لعرض الـ AboutForm dialog. أغلق dialog والبرنامج وارجع الى Delphi.

يمكنك استخدام خطوات مشابهة لبرمجة أمر قائمة يعرض الـ AboutForm. فبدلاً من الـ Button أضف الـ MainMenu component على الـ MainForm واضغط الـ object مرتين لإنشاء أوامر القائمة. اختر أمراً مثل HelplAbout فى نافذة الـ form لإنشاء event handler، وأدخل نفس العبارة التى كتبتها لـ OnClick الخاص لـ Button1.

وطريقة أخرى لإضافة unit داخل unit أخرى هى إضافة uses directive جديد فى قطاع implementation للـ compiled unit. وهذا ما يفعله Delphi تلقائياً اذا أجبت بـ Yes فى الـ error dialog فى الخطوة رقم (٩). وللقيام بهذا يدوياً، احذف اسم الـ About unit، والفصلة التى تسبقها من Main unit uses directive التى قمت بتعديلها فى الخطوة رقم (٩). ضع كلمة implementation (أى تنفيذ) فى نفس هذه الـ unit، وتحت هذا السطر مباشرة، أضف uses directive لإدخال الـ About module. يجب أن يبدو كلا الطرفين كما يلى:

```
implementation
uses About;
```

والاختلاف الرئيسى بين هذا الـ method وإضافة Main uses directive module's فى أعلى تلك الـ unit هو أن البنود الموجودة فى القطاع uses للـ unit تكون خاصة بهذه modules إن استخدام units فى تنفيذ الـ modules يساعد أيضاً فى منع الأخطاء المسماة circular errors، والتى تسببها اثنان من الـ unit تستخدم كلاً منهما الأخرى. وهذا أمر مسموح به إلا أنه يتطلب أن تكون لكل منهما أو uses directive فى قطاع implementation الخاصة بهما، Circular unit references من خلال قطاع الـ uses للـ unit غير مسموح به. ولا يهم عدد الـ units أخرى، فنسخة واحدة فقط من كل unit يتم تحميلها فى الذاكرة. إن قطاع

الـ uses تجعل تعريف متغيرات الـ unit متاحة حتى يمكن للبرنامج أن يرجع للمتغيرات ويستدعى الـ method ويستخدم بنود أدخلت :

جدول (٣-١) Item Templates الجديدة المختارة

Template	الفئة	الوصف
Application	جديد	يبدأ تكوين تطبيق جديد؛ كما هو فى اختيار أمر الـ File|New Application
Component	جديد	يزودك بمعلومات حول component جديد تريد إنشاءه وحتى تعلم الكثير عن البرمجة فى Delphi، فإنك لن تستخدم هذا الـ template
Form	جديد	Form خالية ذات أغراض عامة؛ كما هو الحال فى إختيار أمر الـ File|New Form. استخدم هذا template لإنشاء نافذة رئيسية للبرامج، child windows و custom dialog boxes
Unit	جديد	تبدأ Unit جديد ليس له form تابعة بها. استخدم هذا الـ template لإنشاء units لديها توضيحات، procedures، functions تريد أن تتشارك فيها الـ unit الأخرى
About box	Forms	يعرض ملحوظة حق الطبع، ورقم النسخة، ومعلومات البرنامج الأخرى، دائماً ما يعرض استجابة لأمر الـ Help|About
Dual list box	Forms	قائمتان متداخلتان، ذاتا أزرار لتحويل البنود المختارة من إحدى القائمتين الى الأخرى. إنهما ذاتا فائدة فى الإختيار من بين خيارات ملفات، ومواضع المعلومات الأخرى
Tabbed pages	Forms	dialog box مع أزرار OK، Cancel، و Help، وهى GroupBox area تستطيع حمل components اخرى، ومجموعة من الـ page tab الظاهرة فى الأعلى تتغير فى حالة

		استخدام complex dialogs التي بها controls كثيرة لتتناسب داخل شاشة واحدة، مثل الـOptions dialog. يستطيع المستخدمون اختيار page tabs لرؤية اجزاء اخرى من dialog.
Standard Dialog	Dialogs	يشبه الـ form الحالية، ولكن له ازرار OK و Cancel وهذه الفئة لـ templates قليلة متشابهة تضيف ازرار Help وتضع جميع الازرار فى اسفل أو فى الجانب الايمن من النافذة.
Password	Dialogs	Dialog صغير له مربع ادخال، وازرار OK وCancel استخدمه لطلب كلمة السر من المستفدين. كلمة المرور لا تكون مرئية اثناء الكتابة.

Tip

فكرة: فكرة فى القطاع interface (الى أعلى الـunit) على انه مساحة عامة تكون مرئية للـ unit's الأخرى. فكر فى قطاع implementation (الى الاسفل) على انه مساحة خاصة يحدث فيها برمجة العمل. الخاص بالـ unit. ان الـ unit's تستطيع استخدام هذه العناصر فقط types، والمتغيرات، والـ method، والـdeclarations والتى تظهر فى قطاع الـ interface التى تم ادخالها. وتستطيع الـ unit استخدام العناصر التى تم تعريفها فى قطاع implementation وبصورة عامة، باستثناء تعديل الـunit uses directive، يجب ان تكتب برمجة جديدة فى قطاع uses للـunit's. يقوم Delphi تلقائياً ببرمجة قطاع interface للـ unit وأية تغييرات تقوم بها فى هذا القطاع قد تتعارض مع قابلية الـ Delphi لإنتاج الـcode .

Tip

فكرة: عندما تقوم لأول مرة بحفظ مشروع ذى forms متعددة، يقدم Delphi مجموعة من save-as dialogs-، واحداً لكل modules لم يتم تسميتها بعد. فى احسن الحالات، قد يكون هذا محيراً فى أسوأ الحالات، قد يؤدى بأكثر المبرمجين خبرة ومهارة الى الجنون. وللحماية من التسمية الخطأ للملفات الـ modules والذى قد يحدث دون قصد فى التطبيقات التى لها form متعددة، قم بانشاء كل forms جديدة منفردة. إضف الـ Name والـ

Captionالتابعين لها، واختر File|Save All لتسمية وحفظ modules الجديدة . سوف يستغرق هذا وقتاً اطول لحفظ كل forms جديدة ، ولكن هذا الاسلوب يعتبر اقل مخاطرة من حفظ forms متعددة فى وقت واحد .

إنشاء Object Repository templates:

ان إنشاء الـ form و application templates الخاصة بك تعد طريقة عظيمة للتشارك فى البرمجة أو لتطوير software shells . على سبيل المثال، اذا كان مستخدمى برنامجك يطلبون شاشات إدخال بيانات متعددة تختلف فى نواحى قليلة، فيمكنك اختصار الوقت بإنشاء templates للعناصر المشتركة العامة . ان فرق وضع البرامج يمكنها ايضاً اختصار الوقت والمجهود لتطوير templates للمشروعات الجديدة، و demos و prototypes .

يمكنك حفظ أى form، أو حتى تطبيق بأكمله، كأنه template. ويقوم بهذا عن طريق إضافة form أو تطبيقك فى الـ Object Repository الخاص بـ Delphi. وهى طريق سهلة وبسيطة، وهى عظيمة فى تخصيص Delphi على تصميمات template. إتبع هذه الخطوات لإضافة أى form أو تطبيق للـ Object Repository :

Note

ملحوظة: ان القرص المدمج الخاص بالكتاب لا يحتوى ملفات لما يأتى . استخدم هذه الخطوات لتحويل الـ form، والتطبيقات، الى template .

١- قم بتطوير وحفظ تطبيقك، أو إفتح أى مشروع موجود. إحضر نافذة الـ form المنشورة (اضغط Shift+F12، اذا لزم الأمر، أو اختر امر الـ View|Form) . يجب ان تشاهد نافذة الـ form وليس الـ code editor، لإضافة الـ form للـ Object Repository.

٢- اضغط Alt+F10 أو إضغط مرة واحدة زر الفأرة الايمن بينما يكون المؤشر داخل نافذة الـ form المنشورة. هذا يؤدى لعرض قائمة اختر امر الـ Add To Repository.

٣- لقد توفر لديك Title، Description ، Page (اختر واحدة من القائمة، أو إضف اسم page، جديدة)، ومعلومات Author خيارية. اضغط زر الـ Browse لاختيار ايقونة لعرض template خاص بك، إما واحدة من دليل

Images\Icons الخاص وDelphi، أو واحد، من تصميمك. عندما ترضى عن إدخالاتك، اضغط OK. ان الـ template الآن في الـ Object Repository، ويظهر في الـ New Items dialog عندما تختار امر الـ File|New الخاص Delphi.

اذا كنت تريد إنشاء template لتطبيق بأكمله- وهي طريقة انيقة لتوفير عناصر تطبيقات كنقطة بداية للبرامج الجديدة - إفتح ملف مشروع التطبيق واختر امر الـ Project|Add To Repository. إدخل معلومات واختر ايقونة كما فعلت في الخطوة رقم (٣). يمكنك الآن اختيار هذا الـ template باستخدام File|New.... لتبدأ تطبيق جديد باستخدام العناصر السابقة.

ملحوظة: يقوم الـ object Repository بتخزين معلومات حول template. وتعتبر البيانات الحقيقية وملفات الـ source code هي الاصول. ان إضافة الـ form أو تطبيق للـ Object Repository لا ينسخ الـ source code و الـ form في Delphi، إنه ينشئ مدخل يشير الى الملفات الاصلية. وكنتيجة لذلك، بعد إضافة form أو تطبيق للـ Object Repository، فإن اية تغييرات تقوم بها في الملفات الاصلية تؤثر على template في الاستخدام في المستقبل عن طريق الـ File|New..... والميزة من هذا هو انه يمكنك تعديل، إزالة اخطاء، وتحديث template دون الحاجة لازالة أو إعادة تحميلها في الـ Object Repository.

ان الـ source files للـ templates الافتراضية لـDelphi تكون في دليل الـ Objrepos في الـ Delphi installation الخاص بك. يمكنك عرض هذه الملفات لتعديل الـtemplates ولكن قد تريد الاحتفاظ بنسخ من الاصول. يمكنك ايضاً عرض الـ template entries في New Items dialog باستخدام امر الـ Tools|Repository....... استخدم الـ display الناتج لعرض الـobjects في كل صفحات الـ New Items، لحذف الـ templates (لم تتغير source files)، ولإعادة تسمية page tab.

وفي بيئة مشتركة، مثل الشبكة التي لها واضعي برامج كثيرين يحتاجون الوصول الى application template و الـ form، فإنك تحتاج ان تقيم مسارا

يدعى "Shared Repository Directory." . يتم تخزين هذا المسار فى سجل الـ Windows . لتغييره، استخدم امر الـ Tools|Environment Options ، واختر Preferences page tab . ادخل اسم مسار مشترك حيث يبحث Delphi عن ملف الـ Object Repository، وهو DELPHI32.dro . وهذا المسار غير ذلك المذكور فى الفقرة السابقة . ان المسار الافتراضى هو دليل تركيب bin\ الخاص بـ Delphi وإنسخ ملف DELPHI32.dro الافتراضى فى دليلك المشترك اذا قمت بتغيير اسم مساره .

Note

ملحوظة: ان الـ Delphi control تصل الى ملف الـ Object Repository المشترك باستخدام ملف إغلاق يسمى . DELPHI32.drl . وهذا قد تم صنعه لمنع اثنين أو أكثر من المبرمجين من الكتابة فى الـ Object Repository فى وقت واحد . رغم ذلك، اذا خرج Delphi بشكل غير طبيعى اثناء تعديل الـ Object Repository، قد لا يتم إزالة إغلاق الملف . لإستعادة إمكانية التوصل الكامل للـ Repository، ببساطة إحذف .DELPHI32.drl

حذف Object Repository templates:

اتبع الخطوات التالية لحذف الـ application template أو form من الـ Object Repository :

١- اختر ......Tools|Repository لاحضار نافذة الـ Object Repository .

٢- اختر اسم الـ page التى يتم تخزين الـ template به - إنه نفس الاسم page tab المعروض عن طريق امر الـ ....File|New . ان الـ template لهذه الـ page معروضة فى نافذة الـ Objects .

٣- اختر الـ templates الذى تريد إزالته واضغط زر الـ Delete Object . اجب بـ Yes اذا كنت تريد إنهاء الحذف .

٤- يمكنك ايضاً حذف الـ page كاملة من الـ templates بواسطة زر الـ Delete Page .

ان حذف الـtemplate لا يمحو الملفات الخاصة به. إنهم يبقون فى ادلتهم الاصلية. لاستعادة الـtemplate الذى تم حذفه، قم باضافته مرة اخرى للـObject Repository. قد تحذف الـform template الأساسية خاصة بالـ Delphi ولكن هذه تعتبر Objects مفيدة للغاية ولا يجب ان تحذفها دون سبب وجيه. ان بعض الأمثلة فى هذا الكتاب تشير الى ان واحدة أو اكثر من الـtemplate القياسية متاحة. اذا لم تستطيع إيجاد template مذكوراً، فمن الممكن ان يكون احدهم قد قام بتعديل الـ Object Repository الخاص بك، وعليك إما ان تعيد انشاءه أو تعيد إنزال Delphi .

استخدام الـ Dialog للنافذة الرئيسية:

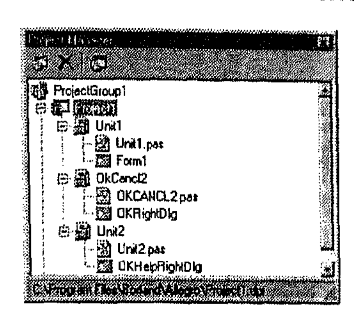
فى بعض الحالات، قد تريد تغيير النافذة الرئيسية لـ forms مختلفة. على سبيل المثال، من المفيد عادة استخدام dialog أو form template اخرى كنافذة تطبيق. وواجهة التطبيق الناتجة تكون نظيفة وبسيطة، ولا يتطلب ذلك menu bar. ويقوم المستخدمون بتشغيل البرنامج بضغط ازرار واختيار controls اخرى فى النافذة.

اتبع هذه الخطوات لتغيير النافذة الرئيسية لتطبيق لتكونform أوtemplate أخرى:

١- ابدأ بتطبيق جديد.

٢- اختر الـ FilelNew......، واختر الـ form template من نافذة الـ New Items dialog، والمعروف أيضاً بالـ Object Repository. أختر Dialog page tab واختر الـ Dialog مع Help form template.

٣- اختر الـ ViewlProject Manager [انظر شكل (٣-٥)]. أبرز الـProject1 entry، واضغط الزر الايمن للفأرة لاظهار القائمة اختر امر الـ Options وفى الـ Project Options dialog الناتج، إختر الـ Forms page tab، وقم بتغيير المدخل فى الـ "Main form" edit box ليكون OKHelpRightDlg (يمكنك ان تقوم بذلك ببساطة باختيار اسم الـobject من القائمة). اضغط OK لقد قمت الآن بتحديد ان الـOKHelpRightDlg يجب استخدامه كالـform الرئيسية للتطبيق.

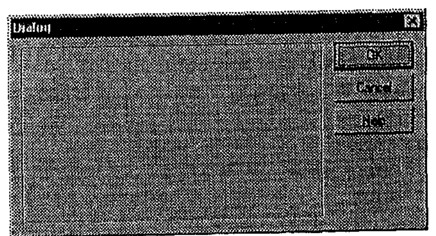


شكل (٣-٥): نافذة الـ Project Manager الخاص بـ Delphi

٤- يجب ان تعرض نافذة الـ Project Manager مرة أخرى. استخدم هذه النافذة لحذف الـ form الرئيسية القديمة للتطبيق، والتى لاحاجة لها الآن. لفعل هذا، قم بابراز الـ Unit1 فى نافذة الـProject Manager ، اضغط زر الفأرة الايمن لتظهر قائمة، واختر Remove من امر المشروع. اجب بـ No اذا حفزك الـDelphi على الاحتفاظ بـ Unit1.pas. هذا يؤدى الى حذف الـ form الافتراضية للمشروع وملف الـsource code unit الخاصة بها. وتختفى الـ form ا القديمة ايضاً من العرض، تاركة نافذة الـ OKHelpRightDlg وملف الـ source code فقط.

٥- اغلق نافذة الـ Project Manager أو إبعدها جانباً. اضغط المشروع باستخدام .....File|Save All و، كما هو معتاد، قم بتسمية الـ main unit والمشروع بـ Test (الاسماء بالضبط لا تهم).

٦- قم بعملية الـ Compile، والـ link وتشغيل التطبيق بضغط F9. يوضح شكل (٣-٦) النافذة الرئيسية للبرنامج - dialog box ذا ازرار Help و Cancel و OK. لانك لم تحدد اية event handler لأزرار الـ dialog، فإن الازرار لا تفعل شيئاً.



شكل (٢-٦): الـ dialog template يستخدم كنافذة رئيسية

إغلاق نافذة:

ان إغلاق نافذة بشكل سليم يعد امراً هاماً تماماً كاظهار واحدة وتشغيلها .

كـمـا عـرفت، فى الـ source code، يمكنك إغـلاق أى Delphi form object باستدعاء الـ Close procedure الخاص بها . اذا كانت النافذة هى نافذة التطبيق الرئيسية، فإن هذا ينهى البرنامج ايضاً . قم بانشاء برنامج event handler بضغط الـ Button مرتين واستدع الـ Close كما يلى :

```
begin
  Close;   { <-- insert this statement here }
end;
```

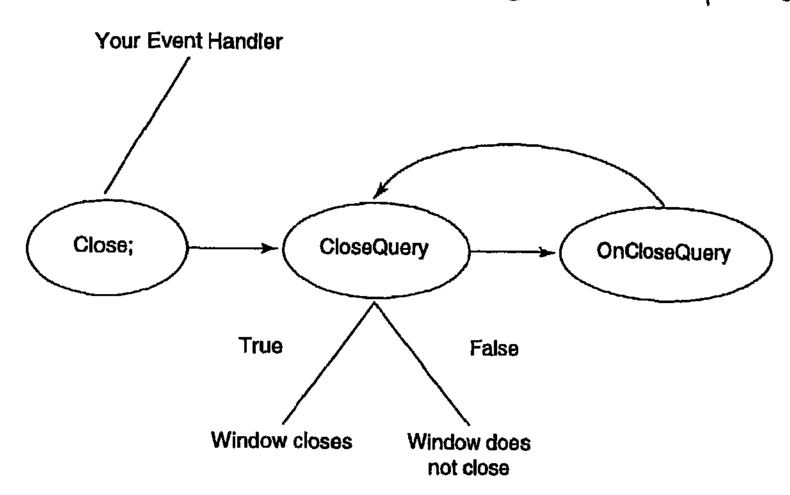
رغم ان هذا يبدو بسيطاً، فإن استدعاء Close يؤدى الى تشغيل اعمال كثيرة خلف الستار. لا يجب عليك برمجة هذه الانشطة- إنها تحدث بشكل تلقائى، ولكن من المهم ان تفهم العملية داخلياً، يقوم الـ Close باستدعاء function أخرى، وهىCloseQuery ، والتى تعود True اذا كان امر إغلاق النافذة مقبولاً . اذا عادت الـ CloseQuery بـ False، فإن استدعاء الـ Close يكون قد تم إلغاءه والنافذة لم تغلق. اذا عادت الـ CloseQuery لـ True، يقوم البرنامج بتنفيذ OnClose (اذا كان هناك واحداً) لهذا الـ form .

ان تسلسل الاغلاق الخاص بـ Delphi يوفر فرص جيدة عديدة لأداء اعمال عندما تغلق الـ form داخلياً، يقوم الـ CloseQuery بتنفيذ الـ OnCloseQuery ، والذى يمكنك استخدامه لتحديد ما اذا كان اغلاق نافذة الـform أمراً آمناً. على سبيل المثال، يمكن للـ OnCloseQuery الخاص لك فحص اذا ما كان المستخدم قد قام بحفظ البيانات الهامة . اذا لم يكن، يمكن للبرنامج ان يعرض تحذيراً، أو يمكنه إلغاء طلب إغلاق النافذة. وشكل (٣-٧) يوضح الاعمال التى تحدث عندما تستدعى الـ Close .

ويوضح مثالاً بسيطاً كيفية استخدام الـ OnCloseQuery لتأكيد نية المستخدم لإنهاء البرنامج باغلاق نافذته الرئيسية . ابدأ تطبيق جديد، ثم اتبع الخطوات التالية :

١- اضف Button component على الـform . اضغط مـرتين على الـ Button واضف Close؛ بين كلمتى البداية والنهاية الرئيسيتين للـevent handler الناتج.

٢- اضـغط F9 لإجراء عمليتى الـ compile و الـ link وتشـغيل البرنامج. اضـغط Button object للانهاء والعـودة لـ Delphi . هذا يعـرض إغلاق form عادى. بالتعمق داخل هذه العـملية، يمكنك إضافة جملة تحذير يسمح للمستخدمين بتغيير آراءهـم بشأن إنهاء البرنامج.



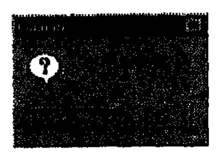
شكل (٣-٧): Delphi يؤدى هذه الاعمال عندما يستدعى البرنامج الـ Close event للنافذة

٣- ومرة اخرى فى Delphi، اختر الـ form بالضغط فى خلفيتها. تأكد ان الـ Button1 objects لم يتم اختـيـاره. اذهب الى الـ Events page فى نافذة الـ Object Inspector، واضـغط مرتين مؤشر الفأرة فى الحقل، القيمة الى اليمين من الـ OnCloseQuery . ادخل البـرمـجـة الموجـودة فى القـائمـة (٣-١). وكمرجع، توضح القائمة الـ event handler كاملاً - فقط ادخل العبارات الواقعة بين كلمتى البداية والنهاية الرئيسيتين. على القرص، تجد هذه القائمة فى دليل الـ OnClose. وتعـرض البـرمـجـة message dialog له ازرار Yes أو No . اذا اختار المستخدم Yes، فإن الـ MessageDlg يرجع القيمة mrYes، وتحدد عبارة if الـ CanClose بـ True. أمـا اذا اخـتـار المسـتـخـدم No، يحـدد البـرنامج الـ CanClose بـ False.

٤- اضغط F9 لتشغيل البرنامج. هذه المرة، عندما تضغط زر النافذة، يطلب message dialog تأكيداً كما هو موضح فى شكل (٣-٨). اختر Yes للإنهاء اختر No لإلغاء طلب الاغلاق.

Note

ملحوظة: إدخل بحذر البرمجة الموجودة فى قائمة (٣-١) بالضبط كما هو موضح، أو إنسخها من ملف الـ CloseQuery.pas على القرص المدمج فى دليل الـ Source\Misc. اذا ادخلت هذه، لاحظ الاقواس المربعة والاقواس الأخرى فى هذا الـ code، والتى قد تبدو متشابهة فى الصفحة المطبوعة.



شكل (٣-٨): message dialog للتحذير

القائمة (٣-١): OnCloseQuery event handler

```
procedure TForm1.FormCloseQuery(Sender: TObject;
  var CanClose: Boolean);
begin
  if MessageDlg('End the program?',
      mtConfirmation, [mbYes, mbNo], 0) = mrYes
        then CanClose := True
        else CanClose := False;
end;
```

Form Frameworks:

تصف الاقسام التالية بعض الأشياء المنظمة التى تستطيع فعلها بالـ form، مثل إظهار scroll bars، جعل نافذة تبقى على قمة نوافذ اخرى وإنشاء الـ splash screen عند البدء.

:Quick-and-dirty scroll bars

ان الـ scroll يعد واحداً من اقدم الوسائل المعروفة لاستعراض المعلومات فى مساحة صغيرة نسبياً. واليوم، اصبح الـ scroll واحداً من حيل العرض المقبولة فى نظم الحاسوب الحديثة لرؤية وثائق ضخمة على شاشات صغيرة. بالإضافة الى كونه اداة واجهة تطبيق مفيدة، يعد الـ scroll bar رابطة بالماضى.

كما تعرف، ان scroll bar هو احد Windows controls التى تضغطه، تسحبه، تعرض المعلومات فى نافذة. اضغط زرار للـ scroll bar ، لتعرض بسرعة بطيئة (سطر واحد فى المرة، مثلاً)، واضغط داخل الـ shaded bar لتمر على البيانات اسرع. اسحب الـ thumb box لتحصل على منظر آخر. على سبيل المثال، لتنتقل الى نقطة المنتصف بالوثيقة، اسحب thumb box الى منتصف الـ Bar.

فى الـ Window 95، تطورت الـ thumb box لتحدد كم المعلومات الغير معروضة فى النافذة – و الـ thumb box كبير تحدد ان معلومات قليلة جداً فقط هى التى لا ترى، بينما يحدد thumb box صغير الى انه هناك معلومات كبيرة خارج نطاق الشاشة. لا يجب عليك اداء أية برمجة خاصة لتشغيل هذه الخاصية.

إضافة scroll bar للـform:

ان إضافة scroll bar لـform يعد أمراً سهلاً، ولكن يمكنك عمل تنقيات عديدة للـscroll بشكل اكثر سلاسة. اتبع الخطوات التالية لإضافة scroll bar لتطبيق الـ BitView من الباب الثانى.

١ – انسخ التطبيق BitView من الباب الثانى فى دليل جديد يسمى Mybits (أو أى اسم آخر). انسخ فقط ملفات Bitview.dpr، و Main.dfm، و Main.pas. لا تنسخ اية ملفات اخرى، خاصة مثل الملفات المكتبية التى تنتهى بـ dsk. والتى قد تحتوى على اسماء مسارات لدليل الـ Bitview الافتراضى. وكمرجع، فإن الملفات التامة لهذا البرنامج توجد فى دليل الـ Bitview 2 على القرص المدمج.

٢ – افتح مشروع الـ Bitview (تأكد من التغيير الى الدليل الذى انشأته فى الخطوة رقم (١)). اضغط مرة واحدة الـ outlined BitImage، والذى يملأ

الـclient area للـ form (أو، إختـر BitImage من القـائمـة بالـ Object Inspector). وفى صفحة الـ Properties فى الـ Object Inspector، اضغط مرتين خاصية الـ Stretch التابعة للـ bitmap لتغيير الـ True الى False. ومع كون الـ Stretch غير عاملاً، لم يعد البرنامج يقوم بتعديل bitmap لتتناسب مع حجم الـBitImage ولذا، فأنت تحتاج الى إضافة scroll bar لترى الصور أكبر من النافذة.

٣- حد خاصية الـ WindowState التابعة للـ MainForm بـ wsNormal، حتى يتم عرض النافذة فى حجمها المصمم بدلاً من الشاشة كاملة. (اختر MainForm من القائمة الخاصة بالـ Object Inspector). وقم أيضاً بتغيير الـ Caption الخـاص بالـ MainForm ليكون Bitmap Viewer مع Scroll Bars.

٤- لإضافة scroll bar أفقى للـ form، اضغط مرتين خاصية الـ HorzScrollBar التابعة للـ form لفتح قيمها الفرعية الخمس. اضغط اسم الخاصية، وليس قيمتها. حدد الـ Range بـ ١٠٠٠ وتأكد من أن الـ Visible فى الـ True (وهى القيمة الإفتراضية). عندما تضعها Enter أو تنتقل الى Property أخرى، يظهر scroll bar أفقى فى نافذة الـ form. اذا لم يظهر، قم بتحديد مدى أكبر.

٥- لإضافة scroll bar رأسى للـ form، أعد الخطوة السابقة مع خاصية الـ VertScrollBar للـ MainForm. وتظهر النافذة النهائية للـ form الآن scroll bars أفقى ورأسى.

٦- اضغط F9 لتشغيل التطبيق الذى تم تعديله. افتح أى ملف bitmap مثل الـ Sample.bmp فى دليل الـ Data. أعد تحديد حجم النافذة، واستخدم الـ scroll bar الخاصة بها لتحريك الصورة الى أعلى وأسفل ويميناً ويساراً.

بشكل عام يمكنك استخدام خطوات أبسط لإضافة scroll bars لأى component يقع خلال الـ client area للنافذة. وفى كل خاصية scroll bar، تأكد من:

- أن القيمة الفرعية Visible تكون True.

● أن القيمة الفرعية Range أكبر من خاصية الـ ClientWidth التابعة للـ from . إن القيمة فيما بين ١٠٠٠ و ٢٠٠٠ تكون مناسبة غالباً .

إن scroll bar تفيد فى الأنماط النموذجية وفى إختيار عمليات التحريك؛ رغم ذلك، للحصول على تحريك أكثر هدوءاً، فسوف تحتاج أن تقوم بعدة إصلاحات للتطبيقات التامة كما هو موضح فيما يلى :

ضبط الـscroll bar:

يتوقع مستخدمى الـ Windows ذوى الخبرة أن تقوم الـ scroll bar بالأداء . وفيما يلى بعض الخطوات التى اذا لم يتم إتباعها بدقة ، فسوف تطلق السخرية من مستخدمى برنامجك :

● يجب ألا تظهر scroll bar حتى يفتح المستخدم وثيقة فى النافذة .

● إن ظهور scroll bar يجب أن يشير الى أن هناك مزيداً من المعلومات متاحة بالتحريك . على سبيل المثال ، فى نافذة نص ، يشير الـ scroll bar الرأسى الى أن الوثيقة تحتوى سطوراً أكثر مما يمكن للنافذة أن تظهره .

● إن إعادة تحديد حجم النافذة ، أو تحميل وثيقة أخرى ، يجب أن يعرض أو يحذف الـ scroll bar حسب الحاجة . على سبيل المثال ، اذا كان تكبير نافذة يؤدى الى ظهور الوثيقة بأكملها ، فيجب أن تختفى الـscroll bar . واذا قمت بتحميل وثيقة كبيرة ، فإن الـ scroll bar ، يجب أن تظهر مرة أخرى اذا لزم الأمر .

● إن وضع الإشارة الخاص بـ scroll bar يجب ان يشير الى موضع الرؤية النسبى فى الوثيقة- أو ، للتحدث بالمصطلحات الفنية ، يجب ان يعكس مدى الـscroll bar حجم الوثيقة .

فى البرنامج التعليمى التالى ، فسوف تقوم بكل التحسينات السابقة لتطبيق الـBitView . توضح القائمة (٣-٢) ملف Main.Pas form unit النهائى من دليل الـ Source\Bitview2 على القرص المدمج .

افتح ملف مشروع الـ BitView الذى تم تعديله من الفصل السابق ، أو ابدأ بنسخ جديد للملفات المشروعdpr ، dfm ، وpas . ثم اتبع الخطوات التالية :

١ - اضـغط مـرتين خـصـائص الـ VertScrollBar والـ HorzScrollBar التابعـة للـMainForm ، وحدد القيم الفرعية Range لكل خاصية بصفر مرة أخرى . وهذا يحذف الـ scroll bar من الـ form وتأكد من انها لن تظهر حتى يفتح المستخدم ملف الـbitmap .

٢- اخـتـر Events page tab التـابعـة للـ MainForm فى الـObject Inspector. اضـغط مـرتين حـقل القـيـمـة المجـاور الـ OnResize لإنشـاء الـevent-handler procedure يسمى FormResize . يقوم البرنامج باستدعاء الـ procedure فى أى وقت يتغير فيه حجم النافذة، بغض النظر عن كيفية حدوث هذا العـمل . إدخل العبـارتين التـاليتين بين البداية والنهاية (واضـاً ارجع للقـائمـة (٣-٢) فى نهاية هذا الفصل اذا أردت مزيداً من المساعدة) :

```
HorzScrollBar.Range := BitImage.Picture.Width;
VertScrollBar.Range := BitImage.Picture.Height;
```

٣- تحدد العبارتان السابقتان خصائص الـ Range فى scroll bar الرأسية والأفـقيـة الخاصـة بالـ form لتكون مساوية لعرض وارتفاع الـ Picture الخاصـة بالـBitImage . ومع هذه التـغييـرات، تعكس scroll bar بصورة تلقائية حجم الصـورة . كـما ستكتـشف، ان هذه التغييرات وحدها لا تكفى لضمـان التحريك الهادى .

٤- اضـغط F9 لتـشغيل التطبيق . افتح ملف bitmap . فى هذه المرحلة لا تظهر اية scroll bar حتى تنتهى من اعادة تحديد حجم النافذة . لعرض scroll bar عند فتح ملف جديد فإن هذا يتطلب تعديل الـ FilelOpen event handler . اترك البرنامج وارجع الىDelphi .

٥- لكى تحدد الـ event handler الصحيح، اختر FilelOpen من نافذة الـ form ولان procedure مازال موجوداً لهذا الأمر، فإن Delphi يضع المؤشر على أول عبارة فى event handler . ادخل السطر التالى بعد العبارة التى تحدد اسم الملف خـاصـيـة الـ Caption بالـ form وهذا يكون داخل اعـمـاق كلمـتى البـداية والنهاية الرئيسية . (راجع القائمة (٣-٢) فى نهاية هذا الفصل اذا أردت مساعدة فى وضع العبارة) .

```
FormResize(Sender);
```

٦- اضـغط F9 لتشـغيل التطبيق . افتح ملف bitmap . تظهـر scroll bar تلقائياً اذا لـم تتناسب الصـورة داخل نـافذة البرنامج . قـم بتغيير حجم النافذة اذا لزم الأمر وقم بتجربة عمل الـ scroll bar .

ان العـبـارة التي اضـفـتـهـا في الخطوة رقـم (٥) تسـتـدعى برنـامج الـ event handler مـوجـود من داخل procedure آخر . في والامثلة الاولى عرفت كيفية تحـديد أكـثـر من events لنفس الـ handler ، مما يؤدى للتـشـارك في الـ code الموجود فيما بين عمليات متعددة . وهنا ، فإنك تستخدم اسلوباً آخر للتشارك في الـ code ، ولكن في هذه الحالة ، فإنك قـمت باستدعاء الـ event handler من داخل آخر يؤدى اعمالاً إضافية- في هذه الحالة ، تحميل ملف bitmap من على القرص .

ونتيـجة للعبارة الجديدة ، فعندما تقـوم بتحـميل وثيـقة الـ bitmap ، يقـوم البـرنامج تلقـائياً بتعـديل مـدى scroll bar ، والذى يؤدى الى ظهـور أو إختـفـاء الـscroll bar حسب الحاجة .

Tip

فكرة: عند حـسـاب حـجم الصـورة ، تأكـد من اسـتـخـدام خـصـائص الـ Width والـHeight الصـحـيـحـة . على سـبـيل المثـال ، إن التـعـبـيـر BitImage.Picture.Height يحصل على إرتفاع bitmap قد تم تحميله داخل BitImage component . والتعبير BitImage.Height يشير الى إرتفاع component object ، وليس الى إرتفاع البيانات داخل الـ object ، وهي القيمة التي تريدها غالباً .

القائمة (٢-٣): Bitview2\Main.pas

```
unit Main;

interface

uses
```

```
    Windows, SysUtils, Messages, Classes, Graphics, Controls,
     Forms, Dialogs, Menus, ExtCtrls;

type
 TMainForm = class(TForm)
    BitImage: TImage;
      MainMenu1: TMainMenu;
      File1: TMenuItem;
      Open1: TMenuItem;
      Exit1: TMenuItem;
      OpenDialog1: TOpenDialog;
      procedure Open1Click(Sender: TObject);
      procedure Exit1Click(Sender: TObject);
      procedure FormResize(Sender: TObject);
     private
    { Private declarations }
     public
  { Public declarations }
     end;

var
 MainForm: TMainForm;
```

```
implementation

{$R *.DFM}

procedure TMainForm.Open1Click(Sender: TObject);
begin
 if OpenDialog1.Execute then
    begin
      BitImage.Picture.LoadFromFile
      (OpenDialog1.Filename);
         Caption := OpenDialog1.Filename;
         FormResize(Sender);
       end;
end;

procedure TMainForm.Exit1Click(Sender: TObject);
begin
 Close;
end;

procedure TMainForm.FormResize(Sender: TObject);
begin
  HorzScrollBar.Range := BitImage.Picture.Width;
```

```
    VertScrollBar.Range := BitImage.Picture.Height;
  end;


  end.
```

الـ ListBox scroll bar السريعة:

تقدم هذا طريقة ابسط لإضافة scroll bars لنافذة . وهذا الاسلوب يفيد بشكل خاص فى عرض القوائم الطويلة . قم بتجربة الخطوات التالية :

١ – ابدأ مشروع جديد واضف ListBox component object من Standard palette الى الـ form

٢ – اختر قيمة خاصية الـ ListBox object's Items واضغط الزر البيضاوى لفتح Delphi's string editor . إدخل ٢٠ سطر نص أو ما الى ذلك (أو إنسخ بعض الاسطر من ملف نص) . اغلق الـ editor . اذا اردت ، استخدم خاصية الـ Font لاختيار حجم ونمط الـ font لنص الـ ListBox .

٣ – حدد خاصية الـ Align للـ ListBox's بـ alClient . هذا يدفع الـ ListBox object لتملأ نافذة الـ form والتى قد تريد إعادة تحديد حجمها فى هذا الوقت .

٤ – اضغط F9 للتشغيل . يظهر تلقائياً scroll bar رأسى عندما تكون النافذة قصيرة جداً بحيث لا تستطيع عرض كل سطور النص فى ListBox control . (لقد أنشأت ايضاً الآن text viewer يسهل استخدامه) .

ان الخطوات السابقة تضيف scroll bars رأسية فقط ListBox component objects . لإنشاء scroll bars أفقى ، ادخل العبارة التالية فى event handler على سبيل المثال ، OnActivate event للـ form .

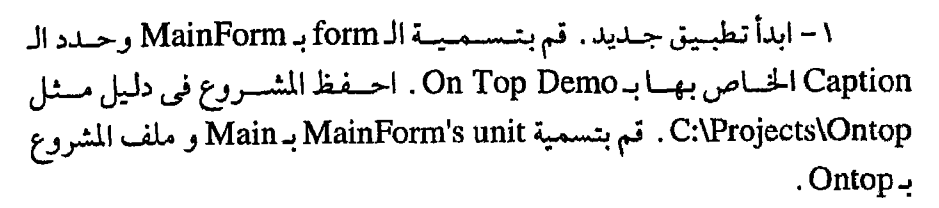
```
ListBox1.Perform(LB_SETHORIZONTALEXTENT, 1000, 0);
```

تستدعى العبارة الـ VCL-internal Perform الموروثة لـ component مثل الـ ListBox من TControl class ويرسل الـ Perform رسالة ، فى هذه الحالة LB_SETHORIZONTALEXTENT ، للـ control . والقيمة ١٠٠٠ هى عرض البيانات فى الـ ListBox مقدراً بالـ pixels وهو بالتقريب هنا .

لقاء نافذة لـ form الخاصة بك على القمة:

ان بعض البـرامج - مـثل السـاعـات، system resource utilities Clacks toolbars- تعتبر اسهل فى الاستخدام اذا بقيت فوق النوافذ الأخرى. وهنا نشرح كيف تحافظ على نافذة الـ form تبقى دائماً فوق الأشياء الاخرى.

على القرص المدمج: ان ملفات البرنامج التعليمى التالى مخزنة على دليل الـ Source\Ontop فى القرص المدمج المرفق بهذا الكتاب.



١- ابدأ تطبيق جديد. قم بتسمية الـ form بـ MainForm وحدد الـ Caption الخاص بها بـ On Top Demo. احفظ المشروع فى دليل مثل C:\Projects\Ontop. قم بتسمية MainForm's unit بـ Main و ملف المشروع بـ Ontop.

٢- اضف MainMenu object فى الـ MainForm. اضغط مرتين ايقونة الـ object وقم بانشاء قائمة تسمى Demo تكون ذات أمرين: Stay on top و Exit. إغلق الـ Menu Designer.

٣- ارجع لنافذة الـ form، اختر امر الـ Exit وادخل Close، بين كلمتى البداية والنهاية الرئيسيتين. اختر امر الـ Stay on top وادخل البرمجة الموضحة فى القائمة (٣-٣). تعد الخطوة الاخيرة من هذا البرنامج التعليمى سوف اشرح كيف تعمل هذه البرمجة. ان نص الـ source code موضوع على القرص فى ملف الـ Main.pas فى دليل الـ OnTop.

٤- اضغط F9 لتشغيل البرنامج الكامل. اختر أمر الـ Stay on top لتجعل النافذة تبقى فوق نوافذ التطبيقات الأخرى. لاحظ ان اختيار الأمر يضيف به علامة صح. اختر الامر مرة اخرى لحذف هذه العلامة.

القائمة (٣-٣): إظهار الـ Stay on top

```
procedure TMainForm.Stayontop1Click(Sender: TObject);
begin
```



```
with Sender as TMenuItem do
 begin
  Checked := not Checked;
    if Checked
      then FormStyle := fsStayOnTop
         else FormStyle := fsNormal;
      end;
end;
```

ان الـ procedure الموجـودة فى القـائمـة (٣-٣) تظهـر إثنين من اسـاليب البرمجة الرئيسية لـ Delphi . ان Sender parameter للـ procedure يمثل شكل الامر الذى اختاره المستخدم . رغم ذلك ، فإن نوع procedure انما هو TObject ، والتى ترتكز عليـهـا كل objects فى Delphi . إذن ، أى object يندرج من الـ TObject يمكن تمريره للـ procedure. لاستخدام هذا الـ object فإن هذا يتطلب ان تخبر Delphi عن كينونته الفعلية .

وهذا قـد تم هنا فى عـبـارة which ، والتى تقـول لـ Delphi ان يعـامل Sender ، كـ TMenuItem object . وهـذا يمكن البـرنامج من الوصـول الى خصـائص امر القـائمة ، واحـد مما هو الـ Checked . ان تحديد هذا الـprocedure بقيمته العكسية مع وجود التعبير not Checked يضع ويزيل علامة صح من القائمة . والاسلوب الثانى الذى تظهره القائمة (٣-٣) هى فى عبارة if التى تحدد الـ FormStyle التابع للـ from باحدى القيمتين - fsStayOnTop أو fsNormal .
وتعتبر نفس خاصية الـ FormStyle هذه متاحة فى نافذة الـ Object Inspector ، ولكن لتجعل النافذة تبقى على القمة استجابة لامر المستخدم ، كما هو موضح هنا ، يجب ان يعين البرنامج بنفسه قيمة الخاصية وقت التشغيل .

وهذا يبرز نقطة هامة . بالإضافة الى تحديد قيم الخاصية باستخدام نافذة الـObject Inspector فى الوقت المحدد ، يمكنك ايضاً تعيين معظم قيم خصائص الـ Object فى وقـت التشـغيـل . علـى سـبيل المثـال ، قم بإنشـاء

الـprocedure event handler، أو أضـف Button object واضـغط له مـرتين .
إضف الجملة بين كلمتى البداية والنهاية :

```
Caption := 'A New Window Title!';
```

عندما تقوم بتشغيل البرنامج ، اضغط الزر لتغيير عنوان الـ form لاحظ ان العبارة يمكن ان تكتب ايضاً بهذا الشكل :

```
MainForm.Caption := 'A New Window Title!';
```

بالرغم من ان هذا يؤدى الغـرض، الا انه يشـيـر مـبـاشـرة لـ MainForm object، مما يجعل الـcode الناتجة اقل فائدة. من ناحية اخرى، إنها اكثر وضوحاً هكذا. ان العبارة الاولى يمكن ان تغير تعليق إية نافذة ، أما الثانية فيمكن ان تغير تعليق الـMainForm فقط . لتغيير نص الـ Button object، استخدم عبارة مثل :

```
Button1.Caption := 'Click Me Again!';
```

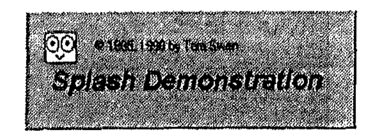
إنشاء الـstartup splash screen :

ان برامج الـ Windows المكتوبة بشكل جيـد تعطى إشـارة بدء ، تسـمى فى بعض الاحيان بـ splash screen . من خلال هذه الشاشة ، يمكنك ان تضيف عامل هام لعرض برنامجك .

ولان المشروع نفسه ينشئ النافذة الرئيسية للبرنامج ، ولان الشاشة يجب ان تظهر قبل ان يحدث هذا ، فإن ملف المشروع يجب ان ينشئ البريق الخاص به . هذا يعنى انك يجب ان تقوم بتعديل الـ source code لملف المشروع ، والتى تعتبر عادة غير مهمة فى برمجة Delphi .

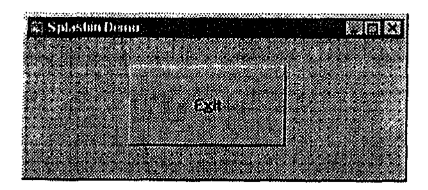
ان ملفات البرنامج التعليمى التالى موجودة فى الدليل الفرعى Splashin . ويوضح الشكل (٣-٩) شاشة البرنامج ، والتى تظهر كنافذة ثابتة الحجم بلا حدود وبلا title bar ، أو قائمة نظام، أو ازرار .

١- ابدأ مـشـروع جـديد . اجـعل اسم الـ form و MainForm وحـدد الـ Caption الخـاص بهـا بـ Splashin Demo . احـفظ المشـروع فى دليل مـثل الـ C:\Projects\Splashin . اجـعـل Main اسـم الـ MainForm's unit و Splashin اسم ملف المشروع .



شكل (٣-٩): شاشة البدء للـ splash screen

٢- اضف الـ Button component على الـ MainForm. قم بتغيير خاصية الـ Name للـ Button لتكون ExitButton و الـ Caption ليصبح Exit. (ملحوظة: اكتب E&xit فى الـ Caption لتصميم الحرف x كمفتاح اختصار. يستطيع المستخدمون عند ئذ ضغط Alt+X لاختيار الزر). ولمجرد الدعابة، قم بتكبير الزر، واعد تحديد حجم نافذة الـ form كما هو موضح فى الشكل (٣-١٠) قم بانشاء الـ OnClick event handler الخاص بالـ ExitButton (يمكنك بسهولة ان تضغط مرتين Button object)، وادخل عبارة الـ ;Close بين كلمتى البداية والنهاية الرئيسيتين للـprocedure



شكل (٣-١٠): نافذة الـ MainForm الخاصة بالـ Splashin'sفى Delphi

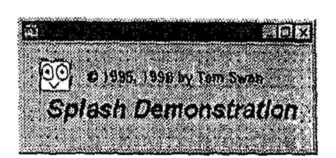
٣- اختر امر الـ File|New Form أو اضغط زر New Form. وهذا يظهر الـ Browse Gallery dialog، والذى يمثل عدة form سابقة البرمجة. (اذا لم تكن ترى الـ dialog، استخدم الـ Options|Environment استخدم Use on New Form check box تحت عنوان Gallery). اختر Blank form templateواضغط Enter أو اضغط OK لإغلاق Browse Gallery dialog.

٤- لقد اضفت لتوك form ثانية للتطبيق قم بتغيير خاصية الـ Name لهذه الـ form لتصبح SplashForm. احذف الـ Caption الخاص بها. قم ايضاً بتغيير خاصية الـ BorderStyle لها لتصبح bsNone وحدد الثلاث قيم الفرعية فى الـ BorderIcons بـ False. بعض هذه القيم اختيارى وربما تود تجربتها مع انماط، نوافذ مختلفة بعد ان تنهى هذا البرنامج التعليمى. يوض الشكل (٣-١١) نافذة الـ

SplashForm فى Delphi التى سوف تنشئها فى الخطوات العديدة القادمة (ان المحتوى الدقيق للنافذة، رغم ذلك، شئ يرجع إليك).

٥- احفظ المشروع. عندما ينبهك Delphi الى اسم ملف الـ unit فتأكد ان الدليل الصحيح هو الحالى (وهو C:\Projects\Splashin اذا كنت تتابع ما نشرح). ادخل Splash كأسم ملف الـ SplashForm's unit.

٦- حدد خاصية الـ Enabled لـ SplashForm بـ False. وهذه تعد إحدى الحالات النادرة التى لا تود فيها ان يتمكن المستخدمون من إعطاء أوامر الفأرة ولوحة المفاتيح لنافذة. فى هذه الحالة يريد ان يكون للبرنامج التحكم الكلى على عرض الـ SplashForm.



شكل (٣-١١): نافذة SplashForm الخاصة بـ SplashIn فى Delphi

٧- قم بإعادة تحديد حجم نافذة الـ SplashForm . ولان النافذة ليس لها outline، اضف الـ Bevel object من فئة الـ Additional. هذا يساعد فى تعريف اطراف النافذة. حدد خاصية الـ Align للـ Bevel1 بـ alClient. وقم ايضاً بتغيير Shape ليصبح bsFrame والـ Style الخاص به ليكون bsRaised. وهذه القيم ترجع لك - جرب تحديدات مختلفة بعد ان تنهى البرنامج التعليمى.

٨- اضف objects component Label والـ Image فى الـ SplashForm. يمكنك ادخال اية عناصر رسومية تريد عرضها اثناء بدء البرنامج. رغم ذلك، لا تدخل اية ازرار أو controls متداخلة اخرى. ويظهر التطبيق نفسه ويزيل splash screen.

٩- هذه الخطوة اختيارية. حدد خاصية الـ SplashForm's FormStyle بـ fsStayOnTop، وقم بتغيير Position ليصبح poScreenCenter. ومرة اخرى هنا، تعتمد التحديدات على رغباتك، ولكن الفكرة الجيدة هى ان تضع splash screen فى المنتصف وان تجعل نافذتها دائماً فوق الأشياء والأخرى.

١٠- اختر View|Project Manager . قم بابراز مشروع الـ Splashin ، واضغط زر الفأرة الايمن لاظهار القائمة . اختر امر الـ Options . وفى الـ Project Options dialog الناتج، اختر Forms page tab . لاحظ ان MainForm و SplashForm موجودان فى قائمة الـ Auto-create forms قم بابراز كل form واضغط زر السهم الايمن لتحريكها الـ Available forms . يتم إنشاء جميع Delphi formsفى الذاكرة فى وقت التشغيل مما يستهلك الذاكرة و system resources فى الحالات المماثلة لهذه، حيث يقوم البرنامج بإنشاء form اثناء وقت التشغيل، يجب ان تزيل هذه الـ form من قائمة الـ Auto-create . إغلق نافذة الـ Project Manager .

١١- بعد ذلك، قم بتعديل الـ code المشروع لعرض الـ splash form قبل ان تظهر النافذة الرئيسية . وهذه واحدة من الحالات النادرة التى تريد فيها ان تدخل عبارات فى ملف المشروع . لعمل هذا، اختر أمر الـ View|Project Source . قم بتعديل العبارات بين البداية والنهاية ليلائم ملف مشروع الـ Splash.dpr فى القائمة (٣-٤) . سوف اشرح المزيد عن هذه البرمجة بعد هذه الخطوات .

١٢- اذا قمت بتشغيل البرنامج فى هذه المرحلة، فإنه يعرض ويزيل startup splash بسرعة كبيرة بحيث لا تكون لديك فرصة ان تراه . لكى يظهر الـ dialog مرئياً لعدة ثوان، اختر الـ MainForm للبرنامج (ذلك الذى له زر كبير) . قم بانشاء لـ OnCreate select handler للـ form . قبل كلمة البداية الرئيسية، اضف متغير longInt يسمى GetTickCount . بين البداية والنهاية، اضف العبارتين لإستدعاء الـWindows GetTickCount function ليحدد الـ Time الحالى لعدد الثوانى الذى يقوم الـ Windows بالتشغيل وعبارة الـ while التى تتأخر لاربعة ثوان إضافية . (يرجع الـ GetTickCount عدد الملى ثانية الذى يقوم الـ Windows بتشغيلها . ان قسمة هذه القيمة على ١٠٠٠ يحولها الى ثوان) . إجع للقائمة (٣-٥)، Main.pas، لترى هذه العبارات .

١٣- أضغط F9 للـ compile وللـ link وللتنفيذ، سوف يعرض البرنامج الـstartup splash dialog لعدة ثوان قليلة ويختفى الـ dialog وتعرض الـ main window .

القائمة (٣-٤): Splashin\Splashin.dpr

```
program Splashin;

uses
 Forms,
   Main in 'MAIN.PAS' {MainForm},
   Splash in 'SPLASH.PAS' {SplashForm};

{$R *.RES}

begin
 SplashForm := TSplashForm.Create(Application);
   SplashForm.Show;
   SplashForm.Update;
   Application.CreateForm(TMainForm, MainForm);
   SplashForm.Close;
Application.Run;
end.
```

تعرض القائمة (٣-٤)، Splashin.dpr، كيف يُمكن لبرنامج إنشاء form object فى وقت تشغيل. لفعل هذا، إستدع الـ Create method فى object's class، كما توضح هذه العبارة:

```
SplashForm := TSplashForm.Create(Application);
```

تقوم هذه العبارة بانشاء الـ object، وتحديده للـ SplashForm، المعرف في الـ Splash.pas. يوجد الـ object الآن في الذاكرة، ولكنه غير مرئي الآن ولكي يظهر وتحدث محتوياته، إستدع Show method، والـ Update للـ object:

```
SplashForm.Show;
SplashForm.Update;
```

القائمة (٢-٥): Splashin\Main.pas

```
unit Main;

interface

uses
  Windows, SysUtils, Messages, Classes, Graphics, Controls,
    Forms, Dialogs, StdCtrls;

type
 TMainForm = class(TForm)
     ExitButton: TButton;
        procedure ExitButtonClick(Sender: TObject);
        procedure FormCreate(Sender: TObject);
      private
     { Private declarations }
      public
   { Public declarations }
```

```
  end;

var
 MainForm: TMainForm;

implementation

{$R *.DFM}

procedure TMainForm.ExitButtonClick(Sender: TObject);
begin
 Close;
end;

{ The following procedure pauses for a few seconds so
  the Splash dialog remains visible for a predetermined
  length of time. }

procedure TMainForm.FormCreate(Sender: TObject);
var
 CurrentTime: LongInt;
begin
 CurrentTime := GetTickCount div 1000;
```

```
    while ( (GetTickCount div 1000) < (CurrentTime + 4) ) do
        Sleep(1);
  end;


  end.
```

يمكنك استخدم خطوات مشابهة لانشاء وعرض أى form فى وقت التشغيل . رغم ذلك ، تأكد ان الـ form لم يتم إنشاءها تلقائياً . (استخدم امر الـ Options فى الـ Project Manager's للفحص) .

Note

ملحوظة: ان استدعاء الـ Update بعد الـ Show ضرورياً هنا فقط لان البرمجة موجودة خارج الـ main message loop للتطبيق ، والذى يقوم بانتاج رسائل الـ Windows . ولا يتم إذن انتاج paint messages، ولذا فإن استدعاء الـ Update يعد أمراً ضرورياً . وعادة ، يجب عليك ان تستدعى الـ Show لإظهار النافذة .

بعد ان انتهيت من استخدام الـ form object ، قم بإخفاءه من العرض ثم احذف الذاكرة التى يشغلها . وتصبح الذاكرة الحره الآن متاحة لـ objects اخرى . على سبيل المثال ، بعد إنشاء النافذة الرئيسية (والتى تتوقف فى هذه الحالة لعدة ثوان حتى يمكنك رؤية الـ splash form)، يستخدم برنامج الاظهار هذه العبارة للتخلص من splashy dialog :

```
  SplashForm.Close;
```

تظهر القائمة (٣-٥) ايضاً اساليب مفيدة للـ OnCreate event. فى الـ MainForm . اولاً ، يقوم الـ procedure باستدعاء الـ GetTickTime، وهى function فى الـ Window ترجع عدد الملى ثانية التى مرت منذ بدء الـ Windows . وهذه القيمة مقسمة على ١٠٠٠ تساوى عدد الثوانى المنقضية .

لتتوقف بضع ثوانى ، يخصص البرنامج القيمة المقسومة للـ CurrentTime ، وهو متغير من نوع الـ LongInt . ثم يستخدم while loop تعبيراً يبدو معقداً لاختيار ما اذا كان الـ GetTickTime (مرة اخرى مقسوماً على ١٠٠٠) اقل من

الوقت الذى تم حـفظه سـابقـاً زائداً أربعـة . ويسـتـدعى الـ while loop ، الـ Windows Sleep function (القـيـمـة المتـغـيـرة (١) تحـدد عـدد المللى ثانيـة للـ sleep) . وهذا يضمن ان العمليات الأخرى تتابع لها فرصة التشغيل بينما يحتل الـCPU-intensive Loop العرض . بعد اربعة ثوانى ، لا يستمر تعبير while loop حقيقياً . وينتهى الـ procedure . وهذا يثبت ان الوقت يمر بالفعل بالرغم من جهودنا المضنية ان نبطئه - اذا فشلت هذه الحيله ، فلا تخف ، ان العالم مازال بخير ، ولكن ساعة حاسبك الآلى قد إنكسرت .

يتـوقف برنامج الاظهـار فـقـط لأنه لا يجـد مـا يفـعله غـيـر ذلك . اذا كـان الـOnCreate الخاص البرنامج يؤدى عدة بدايات ، قـد لا تحتاج الى إدخال وقفة صناعية . رغم ذلك ، لقد اصبحت الحاسبات تزد أو سرعة كل يوم ، والتعليمات التى تستـغـرق اربعـة ثوان اليـوم قـد تنتـهى فى المسـتـقـبل . للضـمـان ، اسـتـدع GetTickCount لاختيار اذا ما كان على الأقل كمية معينة من الوقت قد مرت قبل إنها event handler . الخاص بالـ splash form .

مقدمة حول الـData Modules:

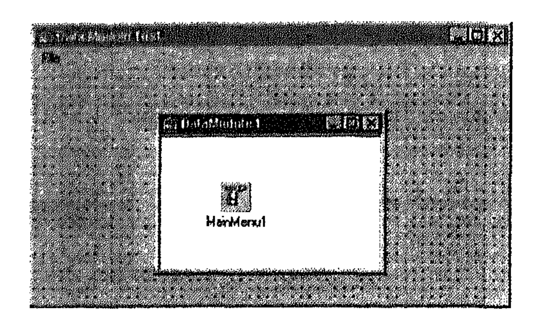
ان data module ، تعـتـبـر form منفـصلة يمكن ان تحـتـوى nonvisual component objects . و Data modulesلا ترى ابداً فى التطبيق التـام - فـهى توفر نوعاً من المخازن للـcomponent التى قـد تود ان تتشـارك فيـهـا مع units و forms اخرى للتطبيق وتعتبر الـ data module ايضاً مكاناً قريباً لحفظ nonvisual objects مـثـل الـ MainMenu والـTimer . يمكنك اسـتـخـدام . data module لتحريك أيقونة الـ object خارج forms أخرى .

Note

ملحوظة: تعتبر data modules مهـمـة بالأخص فى برمـجة قـاعدة البيانات كما هو موضح فى الباب السابع عشر ، تطوير تطبيقات قاعدة البيانات . رغم ذلك ، يمكنك استخدام data module فى أى تطبيق .



على القرص المدمج: اتبع الخطوات التـاليـة لتـعـرف كـيـفـيـة إنشـاء واسـتـخـدام data module . يوضح الشكل (٣-١١) عـرض الوقت المصمم للأمثلة فى Delphi . وتوجد الملفات التامة على القرص المدمج فى دليل الـ DMTest .



شكل (٣-١٢): استخدم الـ data module لتحريك ايقونات مثل الـ MainMenu خارج visible form

١ - إبدأ تطبيقاً جديداً. قم بتسمية الـ form الرئيسية بـ MainForm، وحدد الـ Caption الخاص بها بـ Data Module Test. احفظ كل الملفات، واجعل Main.pas هو اسم MainForm unit واسم مشروعها DMTest.dpr.

٢- اختر File|New Data Module. تظهر نافذة الـ form ذات خلفية بيضاء. يمكنك إدخال nonvisual component objects فقط في هذه النافذة. اذا حاولت ادخال visual component مثل الـ Button، فسوف تظهر لك رسالة خطأ.

٣- اختر File|Save All مرة أخرى. وعندما يظهر أسم الـ Unit2، إدخل الـModule1 واضغط OK. وهذا يجعل اسم source file للـ data module's هو Module1.pas ويعرض ايضاً الـ label فى نافذة الـ code editor. للتطبيق الآن أثنان من الـ forms.

٤- ادخل الـ MainMenu object من الـ Standard palette فى نافذة الـdata module والتى تم تسميتها DataModule1. يمكنك تغيير هذا الاسم فى تطبيقاتك، ولكن لهذا الغرض، فإننا نستخدم الاسم الافتراضى، حسب النظام.

٥- اضغط مرتين الـ MainMenu، قم بإنشاء قائمة File ذات امر Exit. اضغط مرتين امر الـ Exit لإدخال event handler له فى data module's source code إدخل العبارة التالية بين البداية والنهاية:

```
MainForm.Close;
```

٦- أجرى عملية الـCompile بضغط F9. سوف تتلقى رسالة الخطألان الـ DataModule1 تشير الى الـ MainForm، فى Unit اخرى. اما ان تدخل النص باستخدام الـ Main فى قطاع data module's implementation، أو تجيب بـ Yesاذا سألت Delphi ان يقوم بهذا التغيير تلقائياً. (اذا كان باستطاعة Delphi تحديد أى من الـ units ناقصة من قطاع الـ uses، فباستطاعته إدخال اسمائها تلقائياً، ولكنه لن يفعل ذلك دون إذن منك).

٧- يمكنك الآن تشغيل التطبيق، ولكنه لا يعرض الـ MainMenu فى نافذة البرنامج. لفعل هذا، فيجب ان تستخدم Main module وايضاً data module وقطاع uses آخر يمكنه ان يفعل هذه الخدعة. تأكد ان الـ Main module يتم عرضها فى الـcode editor، اختر ....FilelUse Unit، واختر Module1 من القائمة (هناك مدخل واحد فقط). إضغط OK لتضيف جملة uses Module1 فى قطاع Main implementation يمكنك كتابة هذه العبارة يدوياً اذا اردت.

٨- فى اخيراً، اضغط F12 لإظهار نافذة الـ form الرئيسية واضغط بداخلها لعرض قيم خواصها فى نافذة الـ Object Inspector. حدد خاصية الـ Menu بـ DataModule1.MainMenu1 باختيار الاسم من قائمة اللائحة. وهذا يخبر الـ form الرئيسية ان تستخدم الـ nonvisual object، وهو MainMenu، الذى تم تخزينه فى DataModule1 object. اضغط F9 لتشغيل التطبيق التام.

قم بمراجعة الخطوات السابقة بحرص واختبر ملفات الـ source code الموجودة فى القائمة (٣-٦) والقائمة (٣-٧). إنتبه جيداً للقطاع uses فى كل برنامج. ان هذا يعتبر مثالاً لـ circular unit reference والذى، كما ذكرت من قبل، يسمح له فقط قطاع الـ uses من خلال قطاع الـ implementation. لاحظ ان كل unit تشير للاخرى. من الناحية الفنية، ان استخدام الـ data module يتطلب ان تستخدم الـ unit المضيفة فقط (وهى MainForm فى هذه الحالة) لكى تستخدم data module's unit. رغم ذلك، لكى يغلق الـ MainMenu الخاص بـ MainForm module، فيجب ان يستخدم هو الآخر الـ Main's unit. وهذا هو الأمر الطبيعى، و data module غالباً ما تستخدم هذا الاسلوب الدائرى.

Tip **فكرة:** يمكنك ادخال الـ data module في الـ Object Repository كما تفعل في أي form اخرى. ولكن تأكد من انك اعطيت الـ module اسم منفرد (فلا تستخدم مثلاً اسمها الافتراضي DataModulel). فهذا يمنع تداخل الأسمـاء فيمـا بين الـform والـ modules المتعـددة في الـ Object Repository.

القائمة (٣-٦):DMTest\Main.pas

```
unit Main;

interface

uses
    Windows, Messages, SysUtils, Classes, Graphics,
  Controls,
    Forms, Dialogs;

type
 TMainForm = class(TForm)
    private
  { Private declarations }
      public
     { Public declarations }
      end;

var
```



```
  MainForm: TMainForm;

implementation

uses Module1;

{$R *.DFM}

end.
```

القائمة (٧-٢): DMTest\Module1.pas

```
unit Module1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
Forms, Dialogs,
  Menus;

type
  TDataModule1 = class(TDataModule)
     MainMenu1: TMainMenu;
        procedure DataModuleFormExit1Click(Sender:
```

```
    TObject);
     private
    { Private declarations }
     public
  { Public declarations }
     end;

var
 DataModule1: TDataModule1;

implementation

uses Main;

{$R *.DFM}

procedure TDataModule1.DataModuleFormExit1Click(
 Sender: TObject);
begin
 MainForm.Close;
end;

end.
```



الـSplitter Component :

تحتوى الـ Additional palette على components متعددة تعتبر فريدة فى الـ Delphi. وتعتبر هذه الـcontrols مخصصة وهى إما ان تزيد من الـ controls المعيارية، أو تقدم خدمات من المؤكد انك سوف تحتاجها ولن تجدها فى تصميمات الـ controls الاصلية للـ Windows على سبيل المثال، ان BitBtn، الذى رأيته وهو يعمل وسوف تراه يستخدم مرة أخرى خلال هذا الكتاب، يزيد من استخدام زر الـ Windowsفى الرسوم الجرافيكية للحصول على مظاهر أكثر متعة، ويعطى مزيداً من المعلومات.

وإحدى الـ component فى هذه الـ palette والذى اعتبره الأكثر إفادة هو الـ Splitter (class name TSplitter) استخدم هذا الـ component لإنشاء نافذة متعددة يمكن للمستخدمون تعديلها بالضغط والسحب. كثير من تطبيقات الـ Windows لها split windows مشابهة، وهذا الـ control يعد واحداً من controls الأكثر إفادة. وهو ايضاً يسهل استخدامه. جرب هذه الخطوات:

١- إبدأ تطبيق جديد.

٢- قم باسقاط الـ Memo من الـ Standard palette. على الـ form. حدد خاصية الـ Align لهذا object بـ .alLeft. هذا يؤدى الى محاذاة الـ Memo1 فى مقابلة الحافة اليسرى للنافذة.

٣- اضغط الـ Additional page tab وضع الـ Splitter component على الـform ، الى اليمين من Memo1. يظهر الـ control الى الحافة اليمنى من الـ Memo1 مثل المغناطيس والجديد. لاحظ ان خاصية الـ Align للـ Splitter1 هى الـ alLeft فهو نفس القيمة التى حددت للـ object.

٤- ارجع الى Standard paletteوقم باسقاط Memo على الـ form الى اليمين من Memo1 و Splitter1 حدد خاصية الـ Align التابعة للـ Memo2 بالـalClient. فى اغلب الحالات، لابد ان تحدد قيمة الـ Align للـ panel الأخيرة الخاص للـ split window الى الـ client area لهذا هذا الـ object سوف يملأ الحيز الباقى الى أعلى اليسار عن طريق تحديد أحجام الـobjects الأخرى.



قم بتشغيل التطبيق باستخدام F9 . اضغط وأسحب الـ Splitter bar بين Memo components لإعادة تحديد حجمه .

قم بتجربة بعض الخيارات مع الـ Splitter component . اختر Splitter1 من التطبيق الاختيارى ، وقم بتجربة تحديدات هذه الخصائص :

- **Beveled:** حدد القيمة بـ True أو False للحصول على تعديل بسيط لمظهر الـ splitter bar . وعن نفسى ، فإننى أفضل التحديد الإفتراضى بـ True .

- **Cursor:** يجب أن تترك هذه عادة التحديد الإفتراضى ، وهو crHSplit . لا يوجد سبب هام لتغيير مظهر المؤشر ، ولكن يمكنك استخدام مؤشر مختلف اذا أردت .

- **MinSize:** عين الحجم الأدنى لتحديد أطوال وعروض الـ splitter bar . بالرغم من أن هذه القيمة محددة بـ ٣٠ حسب النظام الإفتراضى ، فهى قيمة لا تعنى شيئاً ويجب تغييرها بموجب تحكم البرنامج- على سبيل المثال ، فى الـ OnCreate الخاص بـ form . والصيغة الجيدة هى تعيين قيمة تساوى جزء من عرض النافذة- ١/٤ عرض الـ client area ، مثلاً . تأكد من تحديث الـ MinSize فى الـ OnResize الخاص بـ Form ، بحيث اذا قام المستخدم بإعادة تحديد حجم النافذة بأكملها ، يتم تعديل الحجم الأصغر للـ pane طبقاً لذلك .

- **ResizeStyle:** استخدم واحدة من الثلاث تحديدات الممكنة : rsLine لعرض خط مستقيم أثناء الضغط والسحب ليظهر أين سيكون انقسام النافذة عندما يترك المستخدم الفأرة ؛ rsNone لعدم عرض أية خطوط ؛ أو rsUpdate لإعادة تحديد حجم splitting objects بإستمرار أثناء الضغط والسحب مثل الـ bitmap التى تستغرق وقتاً فى إعادة رسمها . استخدم الـ rsUpdate فى الـ Memo والـ objects الأخرى التى يمكن إعادة رسمها بسرعة . إننى لا استخدم أبداً الـ rsNone لأنه يعطى إشارة بسيطة إلا لشكل المؤشر بأن عملية إعادة تحديد الحجم جارية .

- **Width:** إننى أفضل الواحد وهو أدنى قيمة هنا ، بالرغم من أن القيمة الإفتراضية هى ثلاثة . ومع العرض واحد والـ Beveled محدد بـ True ، يبدو splitter مثل حافة الـ Memo أو أى object آخر أكثر من كونه control منفصل .

والمستخدمين الذين لديهم مشكلة فى استخدام الفأرة قد يجدون من الصعب التحكم فى splitter bar، وفى تلك الحالة يجب عليك استخدام تحديد أكبر أو جعله اختيارياً وتعين خاصية الـ Width فى وقت التشغيل بعبارة مثل :

```
procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Splitter1.Width := StrToInt(Edit1.Text);
end;
```

يمكنك تقسيم النوافذ رأسياً وكذلك أفقياً- إن الأمر يكمن فى كيفية تحديد خصائص الـ Align للـ object . لتجربة هذا، اتبع الخطوات السابقة ولكن قم بتغيير خصائص الـ Align للـ Memo1 والـ Splitter1 لتكون alTop . يجب أن تكون خاصية الـ Align للـ Memo2، alClient . قم بتشغيل التطبيق، واضغط واسحب Splitter bar لإعادة تحديد حجم عرض الـ Memo objects .

Tip **فكرة:** إن خاصية الـ Height للـ Splitter الأفقى تتحكم فى سمكه. وخاصية الـ Width تتحكم فى سمك الـ Splitter الرأسى :

أفكار للمستخدم الخبير

● فى بعض الأوقات، قد يقرر Delphi أن الـ event handler تم وضعه كمرجع ولكنه غير متوفر. ويطلب اذنا لحذف الـ event . وعادة، عندما يحدث هذا، يجب أن تجب بـ Yes، وخاصة اذا كان الـ event واحداً ممن حاولت التخلص منهم. ولكن اذا كان الـ event واحداً ممن تعتقد أنه لا يجب حذفهم، فقد تكون مسحت الـ event handler بطريق الخطأ. أجب بـ No فى هذه الحالة، واضغط الـ event مرتين (أو اختر الـ event handler من القائمة) لإستعادة الـ procedure .

● استخدم File|Open لفتح ملف pas unit. الخاص بالـ form لفحص برمجته. رغم ذلك، فإن هذا لا يضبط الـ form للمشروع الحالى. لفعل هذا، اختر الـ File|Add to project . ولا تظهر الـ form بطريقة تلقائية. ولتظهرها، اختر

ViewlForms أو استخدم الـ Project Manager ، والذى يمكن التوصل إليه من خلال قائمة الـ View . لإظهار الـ form unit ، اختر ...ViewlUnits .

● اذا كنت على دراية بـ text editors آخرين يعرضون الملفات فى نوافذ منفصلة ، فبإمكانك إختيار نافذة Delphi editor قبل عرض modules أخرى . ولكن لا تفعل هذا . إن Delphi يستخدم نافذة editor واحدة لكل unit modules فى المشروع ، وإغلاق هذه النافذة يغلق كل modules المفتوحة . رغم ذلك ، هذا لا يؤدى الى حذف الملفات - إنه يغلقهم فقط فى نافذة الـ editor . للإنتقال الى modules أخرى ، اختر ViewlUnits أو اختر modules من page tab الخاص بنافذة الـ editor . بسبب أنه يحدد نافذة واحدة لكل modules ، يمكنك فتح كل unit فى المشروع دون ازدحام العرض بعشرات النوافذ ، ويمكنك أيضاً تركها مفتوحة .

● اذا كنت تفضل أن يكون لديك نوافذ editor متعددة ، اختر ViewlNew Edit Window .

● إن خصائص الـ Scroll bar لها قيم فرعية إضافية يمكنك استخدامها فى الظروف الخاصة . لرؤية هذه القيم ، اضغط مرتين خاصية الـ VertScrollBar أو الـ HorzScrollBar للـ form . تحدد قيمة الزيادة عدد الـ unit ويقوم thumb box بتحريكها عندما تضغط أزرار اسهم scroll bar . يحدد الـ Margin كم يبعد موقع scroll bar من حافة الـ form ويمثل الـ Position الموضع الحالى لـ thumb box .

● اذا تم تحديد نافذتين أو أكثر ليستقرا على القمة ، فإنهما يتحولان الى نوافذ واقعة فوق بعضها البعض . فلا شئ باستطاعتك أن تفعله حيال هذا . لا يمكن أن تبقى كل نافذة على القمة طوال الوقت .

● يمكنك إدخال template من الـ Object Repository بثلاث طرق : الـ Copy أو Inherit أو Use . وهذه الخيارات يتم عرضها كالـ Radio عندما تختار الـ ...FilelNew لفتح New Items dialog ، والذى يضع قائمة بمحتويات الـ Object Repository . وكل الخيارات الثلاثة لا تكون متوفرة لكل أنواع templates .

● يقوم خيار الـ Copy بنسخ الـ template فى تطبيقك إن حفظ هذه النسخ إذن ينشىء نسخة منفصلة للبنود، والتغييرات المستقبلية للملفات الأصلية لـ template لا تنعكس على تطبيقك.

● ينشىء خيار الـ Inherit، class مشتقه جديدة للاستخدام فى تطبيقك ويمكنك اضافة برمجة جديدة لها. واية تغييرات فى ملفات template الأصلية يشتمل عليه تطبيقك تلقائياً عندما تعيد عملية الـ compile. وهذا يجعل خيار الـ Inherit أكثر الطرق إفادة ومرونة فى استخدام الـ template - على سبيل المثال لتحديث تطبيق، تقوم بتعديل ملفات الـ template الاصلية ببساطة، ثم تجرى عملية الـ compile جميع التطبيقات التى تستخدم هذه الـ template. ولا يجب عليك إعادة إضافتة الـ template.

● والخيار الثالث والأخير، هو خيار الـ Use، يستخدم ملفات الـ template الأصلية مباشرة. على سبيل المثال، يتم فتح الـ source code لـ form مباشرة فى نافذة code editor وآية تغييرات تقوم بها فى الـ form فى تطبيقك يتم حفظها فى ملفات اصلية للـ template. لاتستخدم هذا الخيار اذا كنت تريد الحفاظ على التصميم الأصلى للـ template استخدمه فقط عندما تريد التشارك فى template مع تطبيقات متعددة. وهذا يعتبر الخيار الأقل مرونة ولكنه مفيد فى template مثل الـ data modules .

المشروعات التى يمكنك تجربتها

(٣-١): قم بتصميم وإضافة ايقونة نظام لتطبيق الـ BitView استخدم الـ Windows Paintbrush للـ Image Editor الخاص بـ Delphi's لإنشاء ملف ايقونة له امتداد اسم ملف ico. . أو، انسخ ملف ايقونة موجود بالفعل من شبكة، أو برنامج مجانى للعرض فقط، أو قرص حقل عام. اختر MainForm، واضغط مرتين خاصية الـ Icon التابعة له. استخدم الـ dialog الناتج لفتح ملف الايقونة الخاص بك. يقوم Delphi بنسخ صورة الايقونة فى ملف الـ exe. التام- لان يجب عليك توزيع ملف ico. لمستخدمى برنامجك.

(٣-٢) : استخدم الـ Delphi's AboutBox form الخاصة بـ Delphi من الـ template . اضف حق الطبع الخاص بك ، اسم الشركة ، والمعلومات- العامة الأخرى . قد تريد ايضاً ان تجعل الـ AboutBox مخصص ، لعرض اللوجو الخاص بالشركة (استخدم Image component) .

(٣-٣) : قم بإنشاء الـ form template للحصول على اجابات بـ Yes أو No للمقترحات المختلفة . على سبيل المثال ، قد يعرض dialog سؤالاً وله ازرار Yes و No . اذا تعثرت ، إرجع لمشروع الـ YesNo على القرص المدمج الخاص بهذا الكتاب . افتح المشروع ، واتبع التعليمات الموجودة بهذا الباب لإضافة الـ YesNoDlg form داخل الـ Object Repository . إجعل اسم الـ form unit بـ .Yesno.pas .

(٣-٤) : متقدم . اكتب text-file lister الخاص بك باستخدام الـ BitView2 كمرشد لك في مكان الـ Image ، استخدم Memo . لا يجب عليك اضافة الـ ScrollBar لنافذة الـ form ، رغم انه باستطاعتك ان تفعل ذلك اذا اردت . رغم ذلك ، من الاسهل ان تختار الـ Memo الخاص بالـ form وتحدد خاصية الـ scroll bars التابعة له بـ ssBoth . اذا تعثرت إرجع الى مشروع الـ ListText على القرص المدمج لهذا الكتاب .

(٣-٥) : استخدام الـ Splitter لإنشاء two view لبرنامج قائمة الملف الخاص بك من مشروع (٣-٤) .

ملخص:

- كل مشروع له form object واحد على الأقل يمثل النافذة الرئيسية للبرنامج . ولكن ، قد يكون للمشروعات forms متعددة .
- الـ form تعتبر component ، ولكنها لا تظهر في component palette . قم بإنشاء form باستخدام مشروع جديد أو باختيار File|New Form.... .

● كل form لها unit module مقابل يضع فى قائمة الـ forms الخاص للبرنامج. يجب ان تختلف خاصية الـ Name للـ form عن اسم الـ unit file (فكرة: لتجعل ملفات واسماء الـ form الخاصة بك صحيحة استخدم Name مثل EntryForm واحفظ الـ source code unit فى ملف يسمى (Entry.pas).

● تعتبر الـ form اكثر من مجرد تمثيل لنافذة البرنامج. ان الـ forms لها ايضاً events وخصائص يمكنك برمجتها لاختيار مواصفات النافذة و لأداء اعمال أخرى.

● ان الـ forms و.app templates تفيد فى إنشاء شاشات عامة، انماط نموذجية، و demos اختر امر الـ ......File|New الاختيار من بين template العديد المتوفرة، أو يمكنك إنشاء الـ template الخاصة بك وإضافتها للـ Object Repository. الخاص بـ Delphi .

● يمكنك تحديد أى form كنافذة رئيسية للتطبيق على سبيل المثال، يمكنك استخدام dialog box للحصول على واجهة تطبيق لتطبيق نظيف وبسيط لا يتطلب menu bar.

● الـ OnCloseQuery يؤكد أو يلغى إغلاق نافذة. استخدم هذا الـ event للمساعدة فى منع فقد البيانات.

● تقوم الـ Scroll bar بتشغيل نوافذ لعرض مزيداً من المعلومات اكثر مما يتناسب داخل حدود النافذة. من السهل إضافة Scroll bar سرية للنوافذ، ولكن فى التطبيقات التامة، يجب ان تقوم بعدة تعديلات للـ Scroll bar الخاصة بنوافذك كما هو موضح فى هذا الباب.

● قم بتعين خاصية FormStyle للـ form بـ fsStayOnTop لتجعل هذه النافذة فوق نوافذ التطبيق الأخرى. يمكنك تعيين هذه القيمة فى الـ Object Inspector، أو اثناء وقت التشغيل.

● يمكن للمشروع ان يكون لهforms متعددة كما يظهر برنامج الـ Splashin. يقوم Delphi عادة بإنشاء الـ objects فى وقت التشغيل، وهى حقيقة قد يكون لها آثار على الذاكرة والـ resources الأخرى فى التطبيقات الكبيرة. لإنشاء للـform object فى ظل سيطرة البرنامج، استخدام الـ Project

Manager لتحديد المشروع، اضغط زر الفأرة الأيمن، واختر أمر الـ Options . فى الـ Forms page ، قم بإزالة الـ form من قائمة form . لم يعد يتم إنشاء الـ form object تلقائياً .

● يمكن للـ data modules ان تحتفظ بـ objects مثل الـ MainMenu . وهذا ينقل الـ component خارجاً ويساعد على تنظيم أفضل للبرنامج . لاستخدام data modules ، إدخل اسم الـ unit فى الـ uses directive فى قطاع الـ implementation للـ form unit يمكن تخزين الـ Data modules فى الـ Object Repository .

● استخدم الـ Splitter لإنشاء نوافذ متعددة يمكن للمستخدمين إعادة تحديد حجمها فى وقت التشغيل بالضغط والسحب Splitter bar .

فى الباب القادم، سوف تبدأ فى تعلم كيفية استخدام الـ Delphi components and forms لتصميم واجهات تطبيق جذابة للمستخدم. سوف تعرف ايضاً كيف تبرمج اثنين من اهم أجهزة الإدخال فى الحاسوب : لوحة المفاتيح والفأرة



الجزء الثانى
واجهة تطبيق المستخدم

محتويات هذا الجزء:

إن إنشاء واجهة تطبيق كفء للمستخدم هو أمر غير يسير . بالطبع ، يمكن لغالبية الناس مع وجود معمل مناسب أن يروا buttons ، check boxes ، حقول الإدخال ، والقوائم ليدركوا شيئاً عملاقاً يشبه تطبيق الـ Windows . ولكن هذا يتطلب عمل شاق وتخطيط حذر لإنشاء واجهات تطبيق جيدة تجذب المستخدمين بدلاً من أن تفزعهم .

إن الأبواب الموجودة فى هذا الجزء تغطى تقنيات و Delphi Components لتجميع واجهات التطبيق الصديقة والعملية . سوف تعرف أدق التفاصيل عن إنشاء واجهات تطبيق جرافيكية للمستخدم، واستخدام Component object مثل القوائم، الأزرار، Toolbars، status panels، lists، scrollbars، dialog boxes ، بالإضافة الى إجراء جسم التطبيق الأخرى .



الباب الرابع
برمجة لوحة المفاتيح والفأرة

محتويات الباب:
- حول الجزئين الثانى والثالث
- على لوحة المفاتيح
- مصائد الفأرة

يوماً ما، ولتأمل ان يكون قريباً، سيكون التحدث مع الحاسوب أمراً طبيعياً كالثرثرة مع الجيران. ولكن حتى يتم عمل التحسينات الكبرى لبرمجيات وآلات التعرف الصوتى، سوف تبقى لوحة المفاتيح والفأرة أداتى الادخال الاساسيتين للتخاطب مع تطبيقات البرمجيات. ان الحاسبات يمكنها ان تتحدث بشكل جيد ولكنها لا تعرف كيف تسمع.

كما يعلم المطورين الخبراء، ان امكانات الادخال الخاصة بتطبيق تعرف حرفياً سهولة استخدامه. بالضغط على مفتاح أو أمر فأرة واحد غير ضرورى بامكانه ان يزعج اكثر المستخدمين خبوة والذين لا يرغبون فى كتابة أو ضغط اكثر من الضرورة القصوى. ولان سهولة الادخال تعتبر امر حتمى لنجاح البرنامج، فان مواد استعمال الفأرة ولوحة المفاتيح تعتبر اماكن مناسبة لبداية التحقيقات للجزء الثانى فى تطوير واجهات تطبيق مستخدم التطبيق مع الـ Visual Component Library الخاصة بـ Delphi.

حول الجزئين الثانى والثالث:

ان الجزء الثانى من هذا الكتاب يغطى اساليب تصميم واجهة تطبيق المستخدم باستخدام الـ Delphi components مثل القوائم، الازرار، والـ objects الاخرى

المتصلة بواجهة التطبيق ويشرح الجزء الثالث موضوعات مختصة بالتطبيق مثل الرسوم الجرافيكية والتشارك فى البيانات، ويغطى objects متقدمة فى الـ Visual Component Library. لمعرفة غالبية هذه المعلومات، يجب ان تعرف كيف تؤدى المهام التالية:

- ابدأ تطبيق Delphi جديد.
- عين خصائص الـ Name والـ Caption لـ form.
- أضف component object على الـ form.
- أضف form جديدة الى التطبيق.
- قم بتعديل خصائص الـ component objects والـ form.
- قم بإنشاء procedure للـ component وللـ form.

اذا كنت لا تعرف أى من البندود السابقة، فكلها معرفة فى الجزء الأول، إقرأ الابواب الثلاثة السابقة وأكمل الى النهاية البرامج المعروضة خطوة بخطوة. تأكد من انك تعى كيفية عمل المهام السابقة قبل الاستكمال.

Components covered :

ان الابواب الموجودة فى الجزئين الثانى والثالث تبدأ بقائمة ووصف مختصر للـ covered components والـ classes المرتبطة بها. وتشير غالبية الابواب ايضاً لـ component اخرى والبرامج التعليمية والأمثلة وقد تم وضع كشف بكافة الـ component التى سوف التعامل معها. ويستخدم هذا الباب components متعددة، ولكنه يركز على الـ TForm class الموضحة فيما يلى:

- Form– تعتبر نوافذ التطبيق لـ objects الـ Form component والذى تم برمجته فى TForm class. ويحتوى الـ Form object على components اخرى مثل ازرار، edit boxes، وعناصر اخرى تنشئ واجهة التطبيق لمستخدم التطبيق.

Note

ملحوظة: يبدأ أى مسمى للـ component's Pascal class بحرف الـ T، مما يعنى نوع البيانات. على سبيل المثال، TButton هو اسم Button component class. فى هذا الكتاب، إننى استخدم اسماء

class فقط عند اللزوم- على سبيل المثال، يتم تعريف متغير وذلك تحديد الـ component class له .

البرامج التعليمية وجداول الخواص:

لإختصار التعليمات الموضحة خطوة بخطوة ولتفادى تكرار الشرح الواضح (مثل كيفية حفظ الـ units وملفات المشروع)، فإن البرامج التعليمية العملية التالية تعتبر مختصرة أكثر منها فى الجزء الأول . بالإضافة الى ذلك، تضع بعض البرامج التعليمية كشف يقيم الخصائص فى جدول . يظهر جدول (٤-١) تنسيق جدول الخاصية المستخدم خلال باقى الكتاب .

جدول (٤-١): عينة لجدول الخواص

Component	*Name*	*Property*	*Value*
Form	MainForm	Caption	Widow Demo
Label	XCoordLabel	Caption Alignment Font.Name Font.Size	X Coordinate taRightJustify Arial 24
Edit	Edit1	Text	

إن السطر الأول من جدول الخاصية يضع قائمة تحتوى على أسماء للـ component objects على سبيل المثال، يشير الجدول (٤-١) الى أنه يجب عليك تغيير خاصية الـ Name للـ form الأصلية للبرنامج ليكون MainForm، وحدد الـ Caption الخاص به بـ Window Demo . يجب عندئذ إدخال الـ component الأخرى المدرجة فى القائمة داخل نافذة الـ MainForm .

On the CD-ROM

على القرص المدمج: كل الـ main-window فى هذا الكتاب تسمى MainForm، الا اذا تم الاشارة الى غير ذلك . على القرص المدمج، ان MainForm unit module، وهى Main.pas، مخزنة فى دليل فرعى يتلائم مع اسم المشروع مثل الـ BitView و OnTop . و form modules الأخرى مسماة بطريقة مشابهة على سبيل المثال، NextForm unit module سوف تسمى Next.pas .

والسطر الثانى من الجدول (٤-١) يشير الى انه يجب عليك إضافة Label component object فى الـ MainForm وتغيير الـ Name الخاص به ليصبح X Coordinate . قم بتــحــــديد الـ Caption الخـــاص بهـــذا الـ object بـ XCoordLabel ، وقيم بتغيير خصائص الـ Alignment والـ Font الى القيم المشار إليها . بالنسبة لحجم ومكان الـ object ، إرجع الى الـ objects الموجودة بالكتاب ، والى الملفات المتوفرة .

تفصل النقطة بين الخاصية والقيمة الفرعية الخاصة بها . على سبيل المثال ، يشير عمود الـ Property فى الجدول (٤-١) الى انه يجب عليك تحديد القيمة الفرعية Name لخاصية الـ Font بـ Arial وقيمتها الفرعية Size بـ ٢٤ نقطة .

والسطر الآخير من الجدول (٤-١) يخبرك بان تضيف Edit component object الى MainForm ، ولكن ان تتركك الـ Name الخاص بالـ object محدد بالقيمة الافتراضية ، وهى Edit1 . والعمود الاخير الحاوى يشير الى انه يجب عليك إخلاء خاصية الـ Text للـ object بابراز وحذف هذه القيمة .

ان جداول الخاصية لا تتضمن التحديدات الواضحة أو الغير هامة مثل قيم طول وعرض النافذة . ولا يذكر الجدول ايضاً مظاهر النافذة مثل الـ Bevels المستخدمة فى مشروع الـ KeyInfo لتنظيم عرض النافذة [راجع الشكل (٤-١)] . إننى سوف اشرح كثير من هذه الاشياء الإضافية خلال الكتاب ، ولكننى لا اريد إهدار مساحة فى شرح اشياء واضحة على حساب معلومات اكثر اهمية .

طبقاً للعرف ، تنتهى غالبية خصائص الـ Name للـ component بـ component identifier . على سبيل المثال ، انك تعرف ان XCoordLabel الموجودة فى جدول (٤-١) هى Label component object . والاستثناء المعتاد لهذه القاعدة هو اسم افتراضى حسب النظام مثل Edit1 ، والذى يعنيه Delphi .

على لوحة المفاتيح:

يمكن لتطبيقات Delphi ان تستخدم two methods اساسين لتلقى مدخلات لوحة المفاتيح . الـ method الاول والاسهل هو ان يستخدم الـcomponent object ، مثل الـ Edit ، الذى يستجيب بصورة تلقائية للضغط على المفاتيح لأغراض اكثر شمولاً ، يمكنك إنشاء procedures فى form تتعامل

مع أى مجموعة من ثلاث events (موجودون فى قائمة الـ form object فى نافذة الـ Object Inspector):

- **OnKeyDown** : يستدعى عندما تضغط أى مفتاح، بما فى ذلك الـ function key والمفاتيح الخاصة مثل Shif ، Alt ، و Ctrl .
- **OnKeyPress**: يستدعى عندما تضغط مفتاح إنتاج الرمز ASCII، بما فى ذلك مفاتيح التحكم.
- **OnKeyUp**: يستدعى عندما تترك أى مفتاح .

من هذه events يتلقى parameter واحد على الأقل، يسمى Key، والذى يمثل مفتاح قد تم الضغط عليه، فى الـ OnKeyDown والـ OnKeyUp، ويحدد نوع الـ Key بـ unsigned Word (أى، موجب فقط) والذى يمثل. virtual key code للـ Windows . فى الـ OnKeyPress، يكون للـ Key قيمة الـ Char والتى تمثل رمز الـ ASCII. رغم ان كلا المتغيرين يسمى Key، الا انهما يمثلان معلومات مختلفة من لوحة المفاتيح. لكل رموز الـ ASCII، virtual key code مقابلة لها، ولكن العكس ليس صحيح - فهناك كثير من الـ virtual key ليس لها نظراء من الـ ASCII.

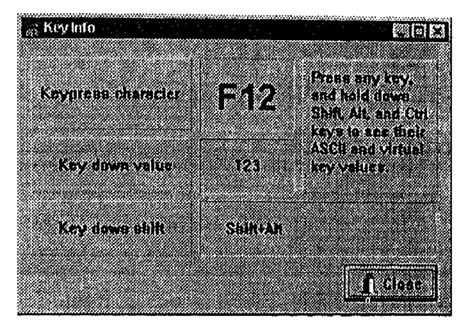
يعبر الـ Windows عن keystrokes لكل virtual key code برمز يسبقه وهو _vk. على سبيل المثال، يعتبر الـ vk_alt code الـ virtual key للـ Alt . لمزيد من المعلومات حول virtual key code، إبحث عن دالات الـ VkKeyScan والـ VkKeyScanEx فى ملف الـ Microsoft Win32.hlp المتوفر مع Delphi. اذا لم يكن لديك هذا الملف أو لم تتمكن من وضع الابعاد (يبدو انها مفقودة فى بعض الطبعات)، ابحث عن الـ Vk فى الـ Windows.pas الموجود فى دليل الـ Source\Win الخاص بـ Delphi .

الاستجابة للـ event الخاصة بلوحة المفاتيح:

يظهر التطبيق، KeyInfo، كيفية برمجة الـ event الخاصة بلوحة المفاتيح الثلاث التابعين للـ form. يوضح الشكل (٤-١) ظهور الـ KeyInfo. قم بوضع وتشغيل هذا البرنامج بتحميل ملف مشروعه من دليل الـ KeyInfo وبضغط F9 .

اضـغـط مـفـاتيح متنوعة، بما فى ذلك الوظيفـة، Ctrl، Shift ، و Alt - ولاحظ القيم المعروضة الى اليمين من بطاقات النافذة:

- **Keypress character**: يوضح هذا الـ label رموز الـ ASCII . وهناك مربع خالى معروض للرموز التى ليس لها رموز مرئية.
- **Key down value**: يوضح هذا الـ label القيمة العددية للمفاتيح. عندما تضغط مجموعة مفاتيح متعددة مثل Shift+Alt+A ، ويوضح هذا المربع قيمة آخر مفتاح تم ضغطه (فهو A فى هذا المثال).
- **Key down shift**: يوضح هذا الـ label حالة الانتقال الحالية. على سبيل المثال، عندما تضغط مفتاحى Alt و Ctrl معاً، يعرض البرنامج Alt+Ctrl، وتوضح قيمة الـ Key down value القيمة العددية للمفتاح الذى تم ضغطه آخراً (وهو Ctrl فى هذه الحالة).



شكل (٤-١): يعرض الـ KeyInfo الـ events الخاصة بلوحة المفاتيح

رغم ان الـ KeyInfo يظهر معلومات لغالبية keypresses ، يستمر البرنامج فى الاستجابة لأوامر لوحة المفاتيح المعيارية. على سبيل المثال، ان ضغط Enter يختار زر الـ Close وينهى البرنامج. وضغط الـ F10 يختار ايقونة قائمة النظام، التى يمكنك بعدها ان تضغط قضيب المسافة أو Enter لفتح القائمة، أو اضغط Esc للعودة للتشغيل العادى.

إنشاء الـ KeyInfo :

إتبع الخطوات المرقمة فى هذا الفصل لإعادة إنشاء التطبيق KeyInfo . يوضح الجدول (٤-٢) قيم خصائص الـ form والـ component الهامة.

جدول (٤-٢): خصائص الـ KeyInfo

Component	Name	Property	Value
Form	MainForm	Caption	Key Info
Label	CharLabel	Caption	
		Alignment	taCenter
		AutoSize	False
		Font:Name	Arial
		Font:Size	24
		Font:Style	[fsBold]
Label	ValueLabel	Alignment	taCenter
		AutoSize	False
Label	ShiftLabel	Alignment	taLeftJustify
		AutoSize	False

نفترض بالتعليمات التالية انك قد قمت بانشاء مشروع جديد، وغيرت الـ Name لـ form النافذة الرئيسية ليصبح MainForm، وقمت بحفظ الـ unit module كـ Main والمشروع كـ KeyInfo لإنشاء التطبيق KeyInfo:

١- أضف الـ Bevels والـ Labels على الـ form لتنظيم النافذة كما هو موضح فى شكل (٤-١). ولا يهم الموضع الدقيق لهذه البنود. استخدام الخاصية الافتراضية Names فى Delphi لهم جميعاً.

٢- بإضافة الـ BitBtnمن الـ Additional. قم بتغيير الـ Kind الخاص به ليصبح bkClose والـ Name الخاص به ليكون CloseBitBtn.

٣- قم بإضافة ثلاث Label's فى منتصف لـ Bevels المفرغة كما هو موضح فى شكل (٤-١). إجعل اسماءها CharLabel، ValueLabel، و ShiftLabel على التوال، واحذف الـ Captions الخاصة بهم. إرجع الى جدول (٤-٢) للحصول على قيم خصائص أخرى. قم بتعديل خصائص الـ Font للتوصل الى عرض مناسب - على سبيل المثال، قد تريد ان تجعل الـ CharLabel Font كبير وسميك.

٤- اختر MainForm، وقم بانشاء الـ OnKeyDown، OnKeyPress، و OnKeyUp. قم بادخال البرمجة الموضحة فى القائمة (٤-١) لهذه الـ procedure الثلاثة. سوف اشرح كيف تعمل هذه العبارات بعد القائمة.

٥- اضغط F9 لإجراء عمليتي الـ compile و الـ link ومن ثم تشغيل المنتج .



على القرص المدمج؛ يمكنك ان تجد source code كاملة لل KeyInfo على القرص المدمج المرفق بهذا الكتاب في دليل الـ KeyInfo's ؛ توضح القائمة (٤-١) ملف Main.pas الخاص به .

القائمة (٤-١)؛ Keyinfo\Main.pasunit Main;

```
interface
uses
  Windows, SysUtils, Messages, Classes, Graphics, Controls,
    Forms, Dialogs, Buttons, StdCtrls, ExtCtrls;

type
    TMainForm = class(TForm)
  Label2: TLabel;
        CharLabel: TLabel;
        Label3: TLabel;
        Label4: TLabel;
        ValueLabel: TLabel;
        ShiftLabel: TLabel;
        Bevel1: TBevel;
        Bevel2: TBevel;
        Bevel3: TBevel;
        Label1: TLabel;
        Bevel4: TBevel;
        Bevel5: TBevel;
        Bevel6: TBevel;
        Bevel7: TBevel;
        CloseButton: TBitBtn;
        procedure FormKeyDown(Sender: TObject; var Key:
    Word;
```

```
      Shift: TShiftState);
        procedure FormKeyPress(Sender: TObject; var Key:
    Char);
      procedure FormKeyUp(Sender: TObject; var Key: Word;
        Shift: TShiftState);
        private
    { Private declarations }
        public
    { Public declarations }
     end;

var
 MainForm: TMainForm;

implementation

{$R *.DFM}

const
 ctrl_A = 1;    { ASCII value for Ctrl+A }
    ctrl_Z = 26;   { ASCII value for Ctrl+Z }

 FunctionKeys: array [vk_f1 .. vk_f12] of string[3] =
  ('F1', 'F2', 'F3', 'F4', 'F5', 'F6', 'F7', 'F8',
      'F9', 'F10', 'F11', 'F12');

procedure TMainForm.FormKeyDown(Sender: TObject; var Key: Word;
 Shift: TShiftState);
var
 s: string;
begin
{- Show integer Key value }
```

```
    ValueLabel.Caption := IntToStr(Key);
{- Show Key shift state }
 s := '';
    if ssShift  in Shift then s := s + 'Shift+';
    if ssAlt    in Shift then s := s + 'Alt+';
    if ssCtrl   in Shift then s := s + 'Ctrl+';
    if Length(s) > 0 then
      Delete(s, Length(s), 1); { Delete final '+' }
        ShiftLabel.Caption := s;
{- Do function key labels }
 if Key in [vk_f1 .. vk_f12] then
      CharLabel.Caption := FunctionKeys[Key]
        else
      CharLabel.Caption := '';  { Erase old character label }
{- Disable Spacebar to prevent selecting Close button }
 if Key = vk_space then
      Key := 0;
end;

procedure TMainForm.FormKeyPress(Sender: TObject; var
Key: Char);
begin
 if Ord(Key) in [ctrl_A .. ctrl_Z] then
      CharLabel.Caption := Chr(Ord(Key) + Ord('A') - 1)
        else
      CharLabel.Caption := Key;
        ValueLabel.Caption := IntToStr(Ord(Key));
end;

procedure TMainForm.FormKeyUp(Sender: TObject; var Key: Word;
 Shift: TShiftState);
begin
```

```
{- Erase the three labels when user releases key(s) }
  CharLabel.Caption := '';
    ValueLabel.Caption := '';
    ShiftLabel.Caption := '';
end;

end.
```

داخل KeyInfo :

ان إثنين من الثوابت، وهما ctrl_A و ctrl_Z في قطاع implementation الـ Main يقرران قيم الـ ASCII لمفتاحي الـ Ctrl+A والـ Ctrl+Z . يستخدم البرنامج هذه القيم لتحديد مدى مفاتيح التحكم بحيث، عندما تضغط ,Ctrl+Q، مثلاً، يوضح الـ label رمز الـ Keypress لهذه القيمة.

ويحدد الثابت الثالث function key strings مثل F1 و F2، ايضاً ليظهر في الـ label رمز الـ Keypress . ومجموعة الثوابت المكتوبة هذه تأتي تلقائياً في البداية قبل القيم الموضحة في الاقواس، وفي هذه الحالة، فهي مفهرسة بـ virtual key codes، vk_f1 الى vk_f12 . وباعطاء variable string اسم الـ FunctionS، تحدد العبارة التالية الـ string مساوياً لـ F3 :

```
FunctionS := FunctionKeys[vk_f3]; {Assign F3 key to
FunctionS}
```

وبرنامج الـ event handler الخاص بلوحة المفاتيح، وهو FormKeyDown، يتلقى قيمة الـ Key وهي تساوى virtual key code لمفتاح مضغوط. أولاً، يعين الـ procedure قيمة Key لخاصية الـ Caption التابعة للـ ValueLabel object من اجل عرض هذه القيمة تحت الـ Label الـ Key down value في النافذة. ولكن لان الـ Captain يعتبر string ، فلا يمكنك تعيين Key مباشرة له. فهذا لن يعمل:

```
ValueLabel.Caption := Key;  { ??? }
```

Note

ملاحظة: خلال هذا الكتاب، استخدمت ثلاث علامات استفهام فى اقواس تعليق، {???}، لأشير الى عمل قابل للتسائل، أو عبارة خاطئة يجب ان تتلافى استخدامها فى برامجك.

بدلاً من ذلك، يجب ان تحول الـ Key الى قيمة string فى هذه الحالة، يمكنك ان تفعل هذا بان تمرر الـ Key الى الـ IntToStr الخاص بـ Delphi، والتى تقبل الأنواع المعرفة بـ Word وتعيد الـ string :

```
ValueLabel.Caption := IntToStr(Key);
```

والعبارة التالية فى الـ FormKeyDown تعين الـ null string للـ s. وهذا ينشيء الـ string لعبارات الـ if القادمة :

```
s := '';
```

Tip

فكرة: لتحويل الـ string الى قيمة عدد صحيح، استخدم الـ StrToInt.

لإنشاء الـ null string، اكتب علامتى تنصيص فردية بلا مسافة بينهما. ان تعيين null string لمتغير الـ string من خلال المعامل =: يمحو الـ string characters.

وتبنى ثلاث عبارات if الـ s string لتوضيح قيم الـ key المجتمعة مثل Shift+Ctrl. ويستخدم العبارات الـ Shift parameter، من نوع الـ TShiftState، وقد تم تمريرها للـ parameter. يعرف الـ Delphi نوع بيانات الـ TShiftState كالاتى :

```
TShiftState =
  set of (ssShift, ssAlt, ssCtrl, ssRight, ssLeft,
    ssMiddle, ssDouble);
```

ومتغير مثل Shift- مجموعة من قيم الـ TShiftState- يمكن ان يحمل قيم صفر، واحد، أو أكثر. على سبيل المثال، يمكنك تعيين هذه المجموعة لـ Shift لتمثيل مفتاحى الـ Shift و الـ Ctrl :

```
Shift := [ssShift, ssCtrl];
```

يعرف Pascal ما يسمى set بـ single bits ، لذا تعتبر هذه الطريقة فعالة لتخزين عناصر متعددة فى مساحة صغيرة للغاية .

وتحدد سلسلة من عبارات الـ if أى من القيم الممكنة (ان وجد) تعتبر فى الـ Shift set . اختبر أولى هذه العبارات :

```
if ssShift in Shift then s := s + 'Shift+';
```

هذا يعنى انه اذا كانت القيمة ssShift فى مجموعة الـShift ، فإن البرنامج يضمShift +string الى نهاية المتغير s . وعبارات مشابهة . تفحص قيم مجموعة اخرى ، وتضم string إضافية الى الـ s .

بعد تكوين الـ string ، يكون المتغير s إما خالياً ، أو يصنع كشفاً باسماء مفاتيح خاصة متنوعة ، تفصلها علامة زائد . اذا لم يكون الـ string خالياً ، فإنه يتنهى بعلامة زائد إضافية ، والتى تحذفها هذه العبارة :

```
if Length(s) > 0 then
  Delete(s, Length(s), 1);
```

ويقوم Length procedure فى الـ Pascal بتحديد الطول لأى string . اذا كان هذا الطول اكبر من صفر ، تستدعى الـ if هنا Delete لحذف الرمز الأخير . للحذف ، قم بتمرير الـ string فى الاتجاه الذى يبدأ منه الحذف ، وعدد الرموز التى ستحذفها من هذا الموضع ولان الرمز الأول فى الـ string يوجد . فى الموضع 1- بمعنى آخر ، فى فهرس الـ string 1- فإن تمرير طول الـ string للـ parameter الثانى لـ Delete يحذف الرمز الأخير فى الـ string .

لعرض الـ string ، يعينه البرنامج لخاصية الـ Caption للـ ShiftLabel :

```
ShiftLabel.Caption := s;
```

للتعامل مع function keys ، تختبر عبارة if أخرى اذا ما كان الـ Key فى مجموعة . virtual key codes من الـ vk_f1 الى vk_f12 . اذا كان كذلك ، يعين البرنامج احد الـ strings الثابتة للـunit من مجموعة الـ FunctionKeys للـ Caption الخاص بـ CharLabel . وان لم يكن كذلك ، يحدد البرنامج هذا الحقل بالـ null string ، بمسح أى نص موضح فى هذه المساحة .

ويعين الـ FormKeyDown ايضـاً null string لـ CharLabel.Caption لمسح أى رمز معروض فى هذا المجال :

```
CharLabel.Caption := '';
```

واخيـراً، فى الـ FormKeyDown، تختـبر عبـارة if اذا مـا كـان الـ Key يساوى مسافة الـ vk_space . لقد اضفت هذه العبارة لأن ضغط قضيب المسافة يختـار الـ control object الحـالى- فى هذه الحـالة، CloseButton . لاحظ ان متغير الـ Key فى رأس (أى السطر الأول من) الـ FormKeyDown مسبوق بـ var . هذا يعنى أن parameter متغير وتعيين قيمة له يمرر هذه القيمة لإستدعاء الـ FormKeyDown . لذا، فإن تحديد الـ Key مساوياً لصفر ليوقف عمل هذا المفتاح (لأن الصفر لا يمثل مفتاحاً بعينه) :

```
if Key = vk_space then
  Key := 0;  { Disable SpaceBar }
```

الـ two procedures التاليان يسهل فهمهما نسبياً . يعين الـ FormKeyUp assigns null لخـصـائص الـ Caption فى الـ CharLabel، ValueLabel، و ShiftLabel؛ وهذا يمسح تلك الـ labels عندما تترك المفتاح .

ويتلقى الـ FormKeyPress قيمة Key والتى تساوى رمز ASCII . اذا كانت قيمة الـ Key الموجودة فى مجموعة القيم المحددة بواسطة مدى الـcontrol-code، من الـ ctrl_A الى ctrl_Z، عندئذ يحـدد البـرنامج الـ Caption الخـاص بـ CharLabel بـ 'A^'، 'B^'، وما الى ذلك . (وغالباً ما تستخدم علامة ^ كرمز لمفتاح تحكم) . وهذا ضرورياً لأن control code ليس لها رمز مرئى مرتبط بها .

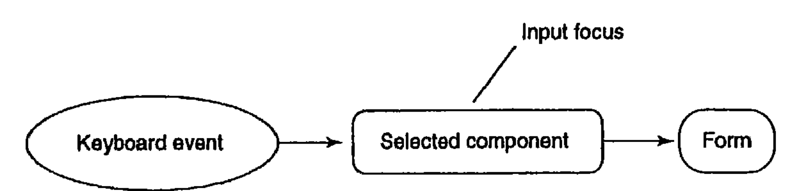
وتعالج عبـارة الـ noncontrol characters else ببساطة بتعيين الـ Key مباشرة الى الـ Caption الخاص بـ CharLabel . لاحظ أنه بإمكانك تعيين متغير Char مثل Key لـ string مثل الـ Caption واحدة من المساحات القليلة التى يكسر فيها Pascal قواعده القوية .

وأخيـراً، يحـدد البـرنامج حـقل الـ Caption الخـاص بالـ ValueLabel بمساواته للـ string مع القيمة الخاصة بالـ Key . عندما تضغط Q، مثلاً، تعين هذه العبارة الـ string 81 للـ Caption .

مفهوم Keypresses:

يستخدم تطبيق الـ KeyInfo، event handler في الـ form لتستجيب لعملية الضغط على المفاتيح. وتستجيب أيضاً كثير من الـ components لنشاط لوحة المفاتيح، ومن الضروري عادة أن تحدد اذا كان الـ component أو الـ form يجب أن يتلقى الـ event الخاص بلوحة المفاتيح.

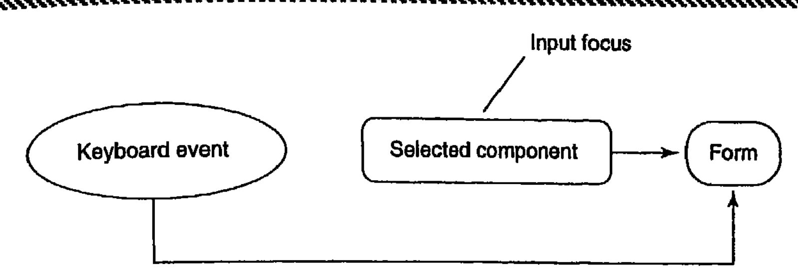
يستخدم الـ Windows، الـ input focus لتحديد أين يرسل الـ event الخاصة بلوحة المفاتيح. على سبيل المثال، الـ component الذي يتم إختياره حالياً في form وله input focus. ولذا، فإنه يتلقى كل الـ event الخاصة بلوحة المفاتيح. اذا لم يتعامل الـ component مع الـ events، مثلاً، لأنه لم يتم برمجته على استخدام مفتاح معين مثل الـ Esc أو Ctrl، فإن هذا الـ events يتم تمريره للـ component الذي يتم التعامل معه والذي هو في الغالب الـ form.



شكل (٤-٢): تذهب الـ events الخاصة بلوحة المفاتيح في العادة الى الـ component المختار ثم الى الـ form

لإرسال الـ events الخاصة بلوحة المفاتيح الى الـ from قبل أن يتلقى الـ component المختار هؤلاء الـ events، حدد خاصية الـ KeyPreview للـ form بـ True. على سبيل المثال، قد تقوم بهذا لتعديل مفاتيح معينة أو لمنع المستخدمين من كتابة رموز غير مطلوبة مثل الأرقام أو علامات الترقيم. يوضح الشكل (٤-٣) العلاقة بين الـ component، والـ from، و input focus عندما يكون الـ KeyPreview في الـ True.

وتوضح تجربة بسيطة قيمة خاصية الـ KeyPreview، والتي يمكنك استخدامها لتحديد المدخلات في ext-editing component. إبدأ مشروع جديد، وإتبع الخطوات التالية:



شكل (٤-٣): حدد الـ KeyPreview بـ True لتجد الـ events الخاصة بلوحة المفاتيح في الـ form قبل أن تستلم الـ component المختار هؤلاء الـ events

١- أدخل الـ Edit في الـ form إنك لا تحتاج الى تغيير الـ Name الخاص به أو أي من الخصائص الأخرى.

٢- قم بتغيير خاصية الـ KeyPreview للـ form من False الى True بالضغط مرتين على قيمتها.

٣- غير الى الـ Events page في الـ Object Inspector، واضغط مرتين الـ OnKeypress للـ from لإنشاء الـ event handler. تأكد من أنك فعلت هذا الـ Form1 object، وليس الـ Edit1. اكتب العبارة التالية بين البداية والنهاية لتحويل الحروف الصغيرة الى حروف كبيرة:

```
Key := Upcase(Key);
```

٤- اضغط F9 لتشغيل البرنامج. اذا عرض Delphi الـ file-save dialogs، اضغط الـ unit والمشروع تحت الاسماء الإفتراضية في دليل مؤقت.

٥- اكتب بعض النص داخل Edit1. يقوم البرنامج تلقائياً بتحويل الحروف الصغيرة الى حروف كبيرة سواء قمت بضغط Caps Lock أو Shift، أو لم تضغطها.

٦- اترك البرنامج، حدد الـ KeyPreview بـ False، ثم اضغط F9 والتشغيل. لم يعد البرنامج يحول الرموز الى حروف كبيرة. وهذا يحدث لأنه أصبح الـ Edit1 له الآن input focus بدلاً من الـ form- ولذلك، يتلقى التحكم الـ events الخاصة بلوحة المفاتيح أولاً.

يعتبر الـ KeyPreview نافعاً أيضاً في تحديد أنواع الرموز التي يمكن للمستخدمين إدخالها. على سبيل المثال، لمنع المستخدمين من كتابة مفاتيح أرقام

داخل الـ Edit ، أضف هذه العبارة الى OnKeypress الخاص بالـ form وحدد الـ KeyPreview بـ True :

```
if Key in ['0' .. '9'] then
  Key := Chr(0);
```

وتفحص عبارة if هذه اذا ما كان الـ Key parameter الذى تم تمريره الى الـ event handler موجود ضمن مجموعة الرموز من '0' الى '9' . اذا كان كذلك ، يعين السطر الثانى من العبارة قيمة الرمز صفر للـ Key . ولأن قيمة رمز ASCII صفر ليس لها معنى ، فهذا يمنع أى إدخال فى الـ Edit معدل .

وتستخدم عبارة if السابقة بعض تقنيات Pascal القيمة التى قد تبدو جديدة بالنسبة لك . والتعبير '9' .. '0' يسمى مدى . إنه يتكون من قيمتين منفصلتين بواسطة نقطتين . ويترجم Pascal هذا التعبير كمساو لسلسة الرموز :

'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'

يمكنك أيضاً أن تؤلف نطاقاً ، من قيم أعداد صحيحة على سبيل المثال ، للمدى 8 .. 5 يساوى سلسلة القيم8 ,7 ,6 ,5 . تأكد من ان تفهم ان رموز الـ ASCII وقيم الاعداد الصحيحة ليست متساوية لبعضها . ان النطاق '9' .. '0' يساوى سلسلة الرموز الرقمية ASCII . والنطاق 9 .. 0 (بدون علامات التنصيص) يساوى سلسلة قيم الاعداد الصحيحة من صفر الى تسعة . يمكنك فقط حشو القيم الترتيبية (الممثلة بقيم اعداد صحيحة) فى مجموعة . لا يمكنك ، مثلاً ، ان تنشئ مجموعة من الـ strings ، لان Pascal لا يستطيع تمثيل قيم الـ string باجزاء بت فردية . ان الـ array هو نوع البيانات المناسب لتجميع القيم المعقدة مثل strings ، records ، class objects .

وتوضح عبارة if السابقة ايضاً كيفية تعيين قيم ASCII خاصة لمتغير من نوع الـ Char . ولانه من المستحيل كتابة رمز بقيمة 0 (أو صفر) للـ ASCII ، فنحن فى إحتياج لاسلوب آخر لتعيين هذه القيمة للـ Key . ولان الـ Key parameter يعد متغير من نوع Char ، يمكنك تعيين أى رمز له بعبارة مثل :

```
Key := 'Q';
```

رغم ذلك ، لا يمكنك تعيين قيمة ASCII مباشرة لمتغير Char . فهذا لا يعمل :

```
Key := 81; { ??? }
```

ان القيمة 81 ليست رمزاً، رغم انها تعتبر كقيمة ASCII للـ Q. لا يمكنك تعيين الصفر مباشرة للـ Key:

```
Key := 0; { ??? }
```

ان Pascal صعب الإرضاء بالنسبة لمثل هذه التعيينات لان القيمة المعينة يجب ان تتطابق مع نوع المتغير. والحل هو تحويل القيمة الى نوع البيانات المناسب، والذى يمكنك فعله هنا باستخدام Chr function توضح العبارات التالية ثلاث methods متساوية منطقياً لتعيين رمز الـ Q للـ Key:

```
Key := Chr(81);  { Calls Pascal Chr to convert 81 to a Char
}
Key := Char(81); { "Casts" the value 81 to the type Char }
Key := 'Q';      { Directly assigns the character Q to Key }
```

إنشاء while-key event :

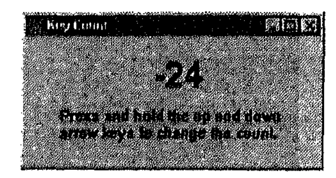
بعض البرامج تحتاج الى أداء اعمال بينما يضغط المستخدمون مفتاح الى اسفل ويستمرون فى الضغط. على سبيل المثال، يمكن ان يستخدم نظام تصميم رسوم جرافيكية اسلوب ضبط ظل لون من الفاتح الى الداكن بينما انت تضغط مفتاح سهم. قدر تطلق لعبة ما بينما انت مستمر فى ضغط قضيب المسافة. ان امكانيات هذه التقنية لاحدود لها، وكما يوضح البرنامج التعليمى التالى، ان عملية البرمجة ليست صعبة.

On the CD

على القرص المدمج: ان واحداً من البرامج الخاصة بهذا الكتاب، وهو KeyCount، يوضح كيفية إنشاء while-key events. لقد تم إنشاء هذا البرنامج فى القائمة (٤-٢)، ويمكن ايضاً أن تجده على القرص المدمج المرفق بهذا الكتاب فى دليل الـKeyCount . رغم ان الـ KeyCount ليس له قيمة عملية، الا انه يزيد ويقلل قيمة العدد الصحيح بينما انت تضغط وتستمر فى ضبط مفاتيح السهام لأعلى أو لأسفل- وبذلك يعرض الاسس الضرورية لإنشاء الـ while-key events.

قم بتشغيل الـ KeyCount. اضغط واستمر فى ضغط مفاتيح السهام لأعلى أو لأسفل لتزيد أو تنقص القيمة الصحيحة فى مركز نافذة البرنامج. اضغط

Alt+F4 للإنهاء يوضح الشكل (٤-٤) مظهر البرنامج. تعطى القائمة (٤-٢) Main.Pas unit الخاصة بالبرنامج.



شكل (٤-٤): عرض الـ KeyCount بعد ضغط مفتاح السهم المشير لأسفل لفترة

القائمة (٤-٢):Keycount\Main.pas

```
unit Main;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes,
Graphics, Controls,
  Forms, Dialogs, ExtCtrls, StdCtrls;

type
  TMainForm = class(TForm)
      Label1: TLabel;
        Label2: TLabel;
        Timer1: TTimer;
        procedure FormCreate(Sender: TObject);
        procedure FormKeyDown(Sender: TObject; var Key:
Word;
    Shift: TShiftState);
          procedure FormKeyUp(Sender: TObject; var Key:
    Word;
        Shift: TShiftState);
          procedure Timer1Timer(Sender: TObject);
     private
```

```
        { Private declarations }
          Count: Integer;
           KeyPressed: Word;
         public
        { Public declarations }
         end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}

procedure TMainForm.FormCreate(Sender: TObject);
begin
  Count := 0;
end; .

procedure TMainForm.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
    if ((Key = vk_up) or (Key = vk_down)) then
     begin
   KeyPressed := Key;
         Timer1.Enabled := True;
        end;
end;

procedure TMainForm.FormKeyUp(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
  Timer1.Enabled := False;
```

```
end;

procedure TMainForm.Timer1Timer(Sender: TObject);
begin
 if KeyPressed = vk_up then
      Inc(Count)
        else if KeyPressed = vk_down then
      Dec(Count);
        Label1.Caption := IntToStr(Count);
end;

end.
```

يوضح البرنامج التعليمى التالى كيفية إنشاء تطبيق الـ KeyCount وكيفية برمجة while-key event . يوضح جدول (٤-٣) قيم الخصائص الهامة للبرنامج.

جدول (٤-٣): خصائص الـ KeyCount

Component	Name	Property	Value
Form	MainForm	Caption	KeyCount
Label	Label1	Caption	0
		Font:Name	Arial
		Font:Size	24
		Font:Style	[fsBold]
Label	Label2	Caption	(enter text from
	Figure 4-4)		
		WordWrap	True
		AutoSize	False
Timer	Timer1	Interval	100

لإنشاء التطبيق KeyCount:

١- لإنشاء النص أسفل نافذة البرنامج ، قم بإضافة الـ Label (Label2 فى جدول ٤-٣)، وقم بتغيير خاصية الـ WordWrap له لتكون True وخاصية الـ AutoSize لتصبح False. قم بتعديل حجم الـ object، ثم اختر خاصية الـ Caption . لا تضغط Enter وأنت تكتب. بدلاً من ذلك، اجعل السطور يحدث لها عملية wrap تلقائياً داخل حدود الـ component والنافذة.

٢- قم بإنشاء global variables، ولقد تم تسميتها بهذا الاسم لأنهما متاحان بشكل عام لكل أجزاء unit module. والمتغير الأول، ويسمى Count، من نوع الـ Integer، يحمل القيمة المقابلة الحالية، المعروضة فى مركز النافذة. والمتغير الثانى، ويسمى KeyPressed، من نوع الـ Word، يحمل قيمة virtual key. قم بتوضيح هذين المتغيرين بالتحول الى نافذة الـ Main.pas بالبرنامج، وأدخل هذه الأسطر بعد الكلمة الرئيسية var (استخدام القائمة ٤-٢ كإرشاد):

```
Count: Integer;
KeyPressed: Word;
```

٣- لكى تجعل العد يبدأ الصفر عندما يبدأ البرنامج التشغيل، اضغط مرتين قيمة الـ OnActivate الخاص بالـ form وأدخل هذا التعيين بين البداية والنهاية:

```
Count := 0;
```

٤- يحتاج البرنامج إثنين من برامج الـ event handlers الاخرى- لضغط المفتاح والآخر لإطلاق المفتاح. إضغط مرتين الأول للـ OnKeyDown الخاص بالـ from وادخل البرمجة من القائمة (٤-٢) بين البداية والنهاية. اقرأ هذه البرمجة بعناية. اذا كانت قيمة الـ Key هى vk_up أو vk_down (الذان يمثلان مفاتيح الاسهم المشيرة لأعلى واسفل) إذن يحفظ البرنامج قيمة الـ Key فى متغير الـ KeyPressed العام. يربط البرنامج ايضاً بالـ Timer1 بالعبارة التالية. بعد هذه العبارة، يبدا العداد باستدعاء OnTimer الخاص به:

```
Timer1.Enabled := True;
```

٥- قم بانشاء event handlers آخر للـ OnKeyUp event الخاص بالـ from. وهذه الـ event يوقف العداد بتحديد خاصية الـ Enabled له بـ False. بعد هذه العبارة، لا يستمرالعداد ويستدعى OnTimer الخاص به:

```
Timer1.Enabled := False;
```

٦- وأخيراً، اختر الـ Timer1 وقم بانشاء الـ OnTimer الخاص به (الـ event الوحيد للـ Timers). والبرمجة لهذا الـ event تزيد أو تنقص الـ Count إعتماداً على ما اذا كان الـ KeyPressed يساوى vk_up أو vk_down. ويعرض الـ event العداد البعد الحالى يتحويل الـ Count الى string من خلال IntToStr وتعيين النتيجة للـ Caption الخاص بـ Label1.

ويمكن لمتغيرات الاعداد الصحيحة الاعداد الصحيحة ان تخزن قيم سالبة وموجبة من -٣٢٧٦٨ الى ٣٢٧٦٨ . ويمكن لمتغيرات الـ Word ان تخزن القيم الموجبة فقط من صفر الى ٦٥٥٣٥ . والمتغيرات من نوعى البيانات والتى تمثل الاعداد الصحيحة فقط ، وليس الكسور .

عندما تحتاج الى إضافة أو طرح واحد من قيمة Ordinal ، استخدم كسور Dec والـ Inc الخاصة بـ Pascal كما يوضح الـ Timer1Timer . على سبيل المثال ، اذا كان الـ Count متغير Integer ، فبدلاً من عبارات مثل :

```
Count := Count + 1;  { ??? }
Count := Count - 1;  { ??? }
```

يمكنك كتابة :

```
Inc(Count);
Dec(Count);
```

ولكن لا يمكنك استخدام الـ Inc و الـ Dec على الخصائص ، والـprocedure له تأثير إيجابى بسيط (ان وجد) على الاداء اثناء التشغيل . فهذه ليست استدعاءات للـ procedure ، رغم انه هذا هو ما يبدو . إنهما فى الحقيقة قد تم تخصيصها لتعليمات الـ CPU التى تزيد أو تنقص قيمة مخزنة فى مكان ما بالذاكرة . ويقوم Delphi optimizer الذكى بإحلال تعبيرات مثل x := x + 1 و y := y - 1 بتعليمات الـ Inc() و Dec() المساوية .

WinTypes، WinProcs، تغيير الـ Enter الى Tab،

رغم ان الـ Windows يتوقع ان يضغط المستخدمون مفتاح الـ Tab للإنتقال من إحد حقول الادخال الى حقل آخر ، كثير من مستخدمون الحاسب يضغطون Enter لهذا العرض . وللأسف ، ان ضغط الـ Enter عادة ما يختار زر OK ، وهو الذى يغلق النافذة أو يحفظ المداخل الحالية قبل ان تتاح لك فرصة إتمامها جميعاً . وهذا يكون محبط للغاية .

وكحل بسيط لهذه المشكلة ، إتبع الخطوات التالية لإعادة برمجة لوحة المفاتيح بحيث ان ضغط الـ Enter يعمل عملاً مشابهاً لضغط الـ Tab :

١- إضف ثلاثEdit component أو أكثر على الـ form ان الـNames الافتراضية ملائمة، ولكن يمكنك تغييرها اذا اردت.

٢- أضف BitBtn وقم بتغيير الـ Kind الخاص به ليصبح bkClose.

٣- حدد خاصية الـ KeyPreview للـ form بـ True.

٤- قم بتغيير الـ ActiveControl الخاص بالـ form ليصبح Edit1 (أو أى Name لأى Edit object).

٥- قم بإنشاء الـ Enter. OnKeyPress event الخاص بالـ form. ادخل البرمجة الموضحة فى القائمة (٤-٣) اضغط F9 للتشغيل، ثم جرب ان تضغط Tab و Enter . تغلق النافذة عند اختيار زر الـ Close. هذا النص موجود على القرص المدمج فى دليل الـKeyMouse، فى ملف Enter2Tab.pas.

القائمة (٤-٣): جعل الـ Enter يتصرف كالـ Tab

```
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
 if Key = #13 then
    begin
   Key := #0;
        SelectNext(ActiveControl, True, True);
      end;
end;
```

تفحص عبارة الـ if فى القائمة (٤-٣) اذا ما كان الـ Key يمثل مفتاح الـ Enter، الذى له قيمة ASCII العشرية. والتعبير 13# يعتبر طريقة اخرى لتمثيل قيمة ASCII الرمزية فى Pascal. اذا كان Key يساوىEnter، فإن عبارة تعيين سوف تحدد الـ Key بـ null. وهذا يبطل الـEnter.

وSelectNext تعتبر method من الـ TWinControl class ، وهى class اساسية لكل الـ controls المرئية التى يمكن ان يكون لها input focus. استدع الـSelectNext لتقديم أو تأخير الـ focus للـ control التالى. ويتم تعريف الـmethod كالآتى:

```
procedure SelectNext(CurControl: TWinControl;
  GoForward, CheckTabStop: Boolean);
```

• **CurControl**: عادة، يجب ان يكون هذا مساوياً للـ ActiveControl الخاص بالـ from. رغم ذلك، قد تخصص control object بدلاً من control مختلف الى الـ control الحالى- على سبيل المثال، لاختيار حقل الـedit الأول فى الـ from لدخول البيانات .

• **GoForward**: حدد هذا الـ parameter بـ True لتنقل الـ focus للـ control التالى. حدده بـ False لتنقل الـ focus للـ control السابق .

• **CheckTabStop** : حدد هذا الـ parameter بـ True لنقل الـ focus للـ control التالى أو السابق فى ترتيب الباب. حدده بـ False لنقل الـ focus للـ control التالى أو السابق بغض النظر عن اذا ما كانت خاصية الـ TabStop للـ control = الـ True أم لا .

إرسال الرسائل:

بدلاً من استدعاء الـ SelectNext، يمكنك ايضاً إرسال رسائل Windows لتغيير الـ focus الى control آخر على سبيل المثال فى القائمة (٤-٣)، تستطيع ان تضع مكان إستدعاء الـ SelectNext العبار التالية، والتى تظهر كيف ترسل رسالة :

```
PostMessage(Handle, WM_NEXTDLGCTL, 0, 0);
```

ترسل العبارة رسالة الـ Windows وهى WM_NEXTDLGCTL والتى تنقل الـ focus الى control التالى فى الصف المرتب حسب tab-order تمثل نافذة الـ form- وهى قيمة يحتاجها الـ procedure للـ Windows المختلفة حتى تعرف أى من النوافذ يجب ان تتلقى رسالة معينة. وقيمتى الصفر لا يتم استخدامهما .

بالرغم من ان هذا الاسلوب يعمل جيداً، الا ان إرسال الرسائل بهذه الطريقة تشكل خطورة فى ان تجعل الـ code غير قابلة للنقل وتعتبر اقل إختصاصاً بالـ object من إستدعاء SelectNext class method. لا يوجد خطأ من الناحية الفنية فى إرسال رسائل الـ Windows للـ controls والنوافذ الأخرى، ولكن عندما يوجد method يؤدى العملية، فمن الأفضل ان تستدعى هذا الـ method .

:Mouse Traps

بعض المستخدمين يتعاملون مع فأرة الحاسب وكأنهم في سباق الانزلاق من على الجبال. وآخرون يتعاملون معها وكأنها تحتاج عجلات للسير. سواء احببتهم أم لا، ان فئران الحاسب موجودة لتستقر، ونجاح برمجياتك قد يعتمد على درجة إجادة برمجية الـ events الخادم بالفأرة الخاصة بتطبيقك، كما سأوضح فيما بعد.

الضغط مرة ومرتين:

لقد إستخدمت بالفعل الـ events الخاصة بضغط الفأرة لأداء اعمال متنوعة. على سبيل المثال، ان تقوم بعملاً اذا ضغط المستخدم زر الفأرة الأيمن، يمكنك ببساطة برمجة الـ OnClick لأى component أو غالبية الـ components. وللإستجابة للضغط مرتين، استخدم الـ OnDblClick يمكنك برمجة أى من الـevents أو كليهما لنفس الـ form أو الـ component وتقع الـ event الخاصة بزر الفأرة الأفتراضى، وعادة هو الزر الأيسر. ولكن، تذكر ان المستخدمين يمكنهم تغيير الزر الافتراضى بالـ Windows Control Panel.

ويعتبر الـ OnClick والـ OnDblClick ملائمتين لكثير، ان لم يكن كل أعمال الفأرة ولكن للتوصل الى تحكم افضل فى الفأرة، يمكنك إنشاء create handlers لهذه الـ events الثلاثة، والتى هى متاحة للـ forms وغالبية الـ components.

- **OnMouseDown**: يستدعى عندما يضغط المستخدم لأى زر بالفأرة.
- **OnMouseMove**: يستدعى عندما يحرك المستخدم مؤشر الفأرة.
- **OnMouseUp**: يستدعى عندما يحرك المستخدم زر الفأرة.

وتتلقى الـ Procedures لهذه الـ events معلومات إضافية حول نشاط الفأرة– على سبيل المثال، أى من الازرار قام المستخدم بضغطه، ومكان مؤشر الفأرة. ولفحص بعض هذه المعلومات، ابدأ مشروعاً جديداً وقم بانشاء الـ event handler خالية لكل من الـ event الثلاثة. إختبر تعريفات الـ Procedure. على سبيل المثال، ها هو تعريف الـ OnMouseDown:

```
procedure TForm1.FormMouseDown(
```

```
Sender: TObject;
 Button: TMouseButton;
 Shift: TShiftState;
 X, Y: Integer);
```

- **Sender**: يمثل الـ object الذى يتلقى هذا الـ event. اذا أشترك أثنان من components أو أكثر فى نفس برنامج الـ event handler، يمكنك استخدام الـ Sender لتحديد أى من الـ object تم ضغطه. (ستعرف المزيد عن هذا فيما بعد).

- **Button**: واحد من القيم الثلاث: mbRight، أو mbLeft، أو mbMiddle. استخدم هذا الـ parameter لتحديد أى من أزرار الفأرة قد قام المستخدم بضغطه.

- **Shift**: مجموعة من الصفر أو أكثر من القيم: ssShift، ssAlt، ssCtrl، ssRight، ssLeft، ssMiddle و ssDouble. استخدم هذا parameter لتحديد اذا ما كان المستخدم يضغط مفاتيح الـ Shift أو الـ Alt أو الـ Ctrl (أو جمع بين هذه المفاتيح) وهو يضغط الفأرة. يمكنك أيضاً استخدام هذا الـ parameter فى تحديد أى من أزرار الفأرة يضغطه المستخدم.

- **x, y**: إحداثى النقطة لموقع الـ form أو الـ component والتى يشير إليها مؤشر الفأرة، استخدم هذه القيم لإختيار الـ objects فى النوافذ، لرسم الجرافيك، ولبداية عملية الضغط والسحب. (بعض النسخ من الـ Delphi documentation تذكر خطأ أن هذه القيم مرتبطة بالشاشة. فى الواقع إنها متصلة بالـ client area فى النافذة، مع الأخذ فى الإعتبار أن الإحداثى 0,0 يمثل الركن الأعلى الى اليسار).

من الأفضل ألا تطلب استخدام زر الفأرة الأوسط، إلا فى الظروف الخاصة عندما تكون واثقاً من أن زر الفأرة الثالث متاحاً. إن أغلب الحاسبات الشخصية لها زران فأرة فقط.

يعرف الـ OnMouseUp event handler نفس الـ parameter مثل الـ OnMouseDown، ولكنه يتم استدعاءه عندما يترك المستخدم زر الفأرة. ويتلقى الـ OnMouseMove event handler أيضاً نفس الـ parameter إلا للـ Button.

للتمييز بين زرى الفأرة الأيمن والأيسر، يجب أن تستخدم عادة الـ Button parameter . على سبيل المثال، لكى تطلق صفيراً عندما يضغط المستخدم زر الفأرة الأيمن أو الأوسط - والذى قد تفعله لتشير الى أن البرنامج لا يميز بينهم - أضف الـ event handler للـ OnMouseDown event الخاص بالـ form وأدخل العبارة التالية :

```
if (Button = mbRight) or (Button = mbMiddle)
  then MessageBeep(0);
```

Note

ملحوظة: إن المصطلحات "أيمن" و "أيسر" يعتبران نسبيين لتحديد مواصفات فأرة نظام المستخدم. تذكر، إن mbLeft يرجع الى زر الفأرة الأيسر، بينما يرجع mbRight الى الزر المقابل. إن المصطلحات "أيمن" و "أيسر" هما بقايا الأيام الخوالى للـ Windows عندما لم يكن ممكناً تحويلهما.

استخدم الـ Shift parameter لتحدد ما اذا كان المستخدم يضغط مفتاحاً أيضاً أثناء ضغط الفأرة. على سبيل المثال، لتجعل الـ speaker يصدر صفيراً اذا ضغط المستخدم زر الفأرة الأيسر أثناء ضغط مفتاح Ctrl، يمكنك استخدام هذه العبارة :

```
if (Button = mbLeft) and (ssCtrl in Shift)
  then MessageBeep(0);
```

إن الجزء الأول من التعبير يعد صادقاً اذا كان الـ button يساوى mbLeft، مشيراً الى ضغطه بالزر الأيسر، والجزء الثانى يختبر اذا ما كان الـ ssCtrl أو أى قيمة ممكنة أخرى موجودة فى مجموعة قيم الـ Shift التى مرت الى الـ procedure.

يمكنك أيضاً استخدام الـ Shift parameter لتحديد أى من الأزرار قام المستخدم بضغطه على سبيل المثال، تنتج العبارة التالية النتيجة المطابقة للعبارة السابقة، ولكن تفحص اذا ما كان الـ ssLeft موجود فى مجموعة قيمة الـ Shift :

```
if (ssLeft in Shift) and (ssCtrl in Shift)
  then MessageBeep(0);
```

يمكنك تكملة هذا الأسلوب للإختبار عن مفاتيح أخرى مثل Alt :

```
if (ssLeft in Shift) and
```

```
    (ssCtrl in Shift) and
      (ssAlt  in Shift)
     then MessageBeep(0);
```

فى مثل هذه الحالات، من الأسهل أن تحدد قيم المفاتيح الممكنة كمجموعة حرفية بين قوسين، وقارنها بالـ Shift :

```
if [ssLeft, ssCtrl, ssAlt] = Shift
  then MessageBeep(0);
```

إن كلا الأسلوبين ليسا متساويين بدقة، على أية حال. إن الأسلوب الأول يجعل الـspeaker يصدر صفيراً اذا كانت الـ ssLeft، و ssCtrl، و ssAlt فى الـShift بغض النظر عما اذا كانت قيم أخرى موجودة فى نفس المجموعة. على سبيل المثال، إن الأسلوب الأول سوف يمكن المستخدمين أن يستمروا فى ضغط مفتاح الـ Shift أيضاً، ولكن الأسلوب الثانى لن يسمح بذلك. وكما يشير هذا، إن المجموعات تعتبر فى متناول اليد، ولكن يجب أن تفكر جيداً فيما تحاول الوصول إليه.

لا تخلط بين الـ Shift parameter ومفتاح Shift. لتحديد اذا ما كان مفتاح Shift مضغوط أثناء الـ event للفأرة، استخدم عبارة مثل :

```
if (ssShift in Shift) then

  ...
```

استخدم الـ x parameter و y لتحديد مكان مؤشر الفأرة أثناء إحداث التحريك، أو الضغط، أو الإطلاق للفأرة. على سبيل المثال، لعرض أحداثيات فأرة عندما يضغط المستخدم زر الفأرة، أضف اثنين من الـ Label component على الـ form وقم بإنشاء OnMouseDown event handler الخاص بالـ form. (تأكد من إختيار الـ form event) أدخل العبارات التالية بين البداية والنهاية :

```
Label1.Caption := IntToStr(X);
Label2.Caption := IntToStr(Y);
```

قم بتشغيل البرنامج واضغط الفأرة داخل النافذة. توضح الـ labels موقع مؤشر الفأرة بالنسبة للـ client area النافذة.



العودة الى الـ sender:

لقد وعدت فيما سبق أن اشرح كيفية استخدام الـ Sender parameter الذى يتم تمريره لكثير من الـ event handlers مثل الـ OnMouseDown . عندما يتشارك objects عديدة في event handler، يمكنك استخدام الـ Sender لتحديد أى من الـ object قد تلقى الـ event .

وكمثال على هذا الأسلوب المفيد، إتبع هذه الخطوات :

١- في نافذة الـ form، أدخل ثلاثة أزرار يحملون الاسماء الإفتراضية وهى Button1، Button2، و Button3. (فكرة: أضف زراً واحداً، اضغط Ctrl+C لتكوين نسخة منه في الحافظة. ثم اضغط Ctrl+V مرتين لإنشاء نسختين آخرتين. هذه الطريقة أسرع من إضافة objects متعددة واحداً تلو الآخر، ولكن عليك أيضاً تعديل الـ Captions الخاصة بالأزرار، وإلا سوف يحملون جميعاً المسمى Button1).

٢- اضغط واسحب الـ rubber band outline لإختيار الأزرار الثلاثة. سوف ترى الأزرار قد تم إختيارهم وتظهر نافذة الـ Object Inspector الآن الـ events والخصائص المشتركة بين الأزرار.

٣- اختر Events page بالـ Object Inspector، واضغط مرتين قيمة الـ OnMouseDown event . هذا ينشيء الـ event handler التى يتشارك فيه الأزرار الثلاثة. بعبارة أخرى، إن ضغط أى زر أثناء وقت التشغيل يستدعى نفس الـ procedure.

٤- من الواضح أن الـ procedure المتشارك فيه يحتاج أسلوباً لتحديد أى من الأزرار قمت بضغطه، هنا يأتى دور الـ Sender. في OnMouseDown procedure، أضف العبارات وتعريف المتغيرات الموجودة في القائمة (٤-٤). على القرص المدمج، يوجد هذا النص في دليل الـ KeyMouse في ملف الـ WhichButton.pas.

٥- اضغط F9 لتشغيل البرنامج. عندما تضغط زراً، تؤكد رسالة أى من الـ button object قمت بإختياره.

القائمة (٤-٤): مثال الـ Sender

```
procedure TForm1.Button1MouseDown(Sender: TObject;
Button: TMouseButton;
 Shift: TShiftState; X, Y: Integer);
var
 s : string;
begin
 if Sender = Button1 then
    s := 'Button1'
     else if Sender = Button2 then
    s := 'Button2'
     else if Sender = Button3 then
    s := 'Button3'
     else s := 'Unknown Object';
   ShowMessage('You clicked ' + s);
end;
```

فى القائمة (٤-٤)، تقوم عبارة الـ if-then-else بمقارنة الـ Sender بـ Button1، Button2، و Button3. وإعتماداً على قيمة الـ Sender، يحدد البرنامج الـ string variable لاسم الزر المناسب. وعبارة sets الأخيرة تحدد المتغير للـ Unknown Object فقط فى حالة أن يكون الـ Sender لا يتناسب مع واحداً من الأزرار الثلاثة.

والعبارة الأخيرة فى القائمة (٤-٤) تستدعى الـ ShowMessage لعرض ملحوظة حول الزر الذى قمت بإختيار. يمكنك تمرير أى ShowMessage. لاحظ كيف أن علامة زائد تضاف للـ string المعين بواسطة عبارة الـ if-then-else السابقة للـ string الحرفى "You clicked".

وعبارة الـ if-then-else فى القائمة (٤-٤) تعتبر عبارة واحدة، وليست عدة عبارات. ولذلك فهى منفصلة عن العبارة التالية لها فصلة منقوطة.

عند استخدام الـ Sender، فإنك غالباً ما تحتاج أن تخبر الـ Delphi أى نوع من الـ object يمثله الـ parameter. يمكنك عندئذ تعيين قيم لخصائص الـ

Sender . على سبيل المثال ، لتغيير الـ Caption لكل زر بعد أن يتم إختياره ، فإن العبارة التالية لا تنفع :

```
Sender.Caption := 'Clicked'; // ???
```

لا يمكنك أن تفعل هذا لأن الـ Sender من نوع الـ TObject ، الذى ينحدر منه كل Delphi component . بمعنى آخر ، إن كل الـ component تعتبر TObject . للتوصل الى خصائص الـ Caption بالأزرار ، باستخدام عبارة with-do لتخبر Delphi إن الـ Sender هو فى الواقع TButton object- أضف العبارة التالية فى الـ procedure الموجود فى القائمة (٤-٤) :

```
with Sender as TButton do
  Caption := 'Clicked';
```

إن تغيير الـ with-do يخبر الـ Delphi أن يتعامل مع الـ Sender كأنه TButton object ، والذى له خاصية Caption . عندما تقوم بتشغيل البرنامج الذى ، تم تعديله ، تتغير كل مسمى للزر الآن لتصبح Clicked بعد أن تختارها . إن استخدام "as" لا يحول الـ Sender الى TButton object- إنه فقط يرى الـ Sender كأنه ذلك النوع الخاص من الـ object . إنها مسئوليتك أن تضمن أن الـ Sender حقيقة هو نوع الـ object الذى حددته . لا يمكنك استخدام الكلمة الرئيسية as لتحويل object الى آخر من نوع مختلف .

مؤشرات الفأرة:

يمكنك تحديد واحداً من الـ objects المتعددة لمخزون المؤشرات لأى form أو component . على سبيل المثال ، ابدأ تطبيقاً جديداً ، وقم بتغيير خاصية الـ Cursor للـ from من crDefault الى crCross . اضغط F9 للتشغيل ، وحرك مؤشر الفأرة فوق نافذة برنامج وبعيداً عنها . يتغير المؤشر تلقائياً الى شكل علامة الزائد ، ثم يعود الى شكل السهم عندما تحركه بعيداً عن الـ client area بالنافذة .

قد يكون للـ Components أيضاً شكل مؤشرات مرتبطة بها والتى تشير الى أنواع العمليات التى يمكن للمستخدم أن يختارها أو يؤديها . على سبيل المثال ، أضف GroupBox component الى الـ form وحدد خاصية الـ Cursor لها بـ crHSplit . اضغط F9 والتشغيل . عندما تحرك مؤشر الفأرة داخل الـ GroupBox

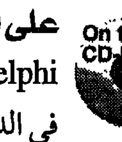
يتحول شكل علامة الزائد الى شكل split-cursor ، والذى يظهر عند حدوث عملية الـ clicking أو حدوث الـ dragging لتقوم الفأرة بتغيير الأحجام للـ split window.

Custom mouse cursors:

يمكنك أيضاً إنشاء أشكال المؤشر الخاصة بك لأى form أو component وهذا الأمر ليس صعباً، ولكن يتطلب بعض العمل الزائد وقليل من البرمجة.

١ - قم بتشغيل الـ Image Editor الخاص بـ Delphi من قائمة الـ Tools لإنشاء أو تعديل صورة المؤشر، وهى عادة لـ bitmap 32×32 pixels ذات لونين.

٢ - احفظ صورتك فى ملف له إمتداد cur.، يمكنك أيضاً أن تجد العديد من ملفات المؤشر على bulletin boards أو الـ public domain و shareware. وكذلك عليك أن تفحص الملفات فى دليل الـ Images\Cursors الخاص بـ Delphi.



على القرص المدمج: اذا لم تكن قد قمت بتركيب ملفات المؤشر الخاصة بـ Delphi وليس لديك وقتاً لإنشاء أخرى، استخدم ملف الـ Sample.cur فى الدليل الفرعى Data على القرص المدمج المرفق بهذا الكتاب.

٣ - يجب عليك بعد ذلك أن تقوم بإدخال المؤشر كـ Resource فى تطبيق Delphi. يتم تخزين الـ Resource داخل ملف الـ exe. الخاص بالبرنامج، ولكن أثناء التطوير، من الأفضل أن تقوم بتخزينهم منفصلين فى ملفات res.. إم عملية الـ Compiling والـ linking للبرنامج يربط ملف بيانات الـ res. فى ملف الـ code- لا يجب عليك توزيع ملفات الـ res. للمستخدمين النهائيين.

٤ - استخدم الـ Image Editor الخاص بـ Delphi بإنشاء ملف resource، من الممكن أن يسمى Cursor.res. استخدم File|New لإنشاء ملف resource جديد، ثم اضغط يميناً داخل نافذة resource لإنشاء bitmap، مؤشر، وأيقونة تكون resource objects. وهذه الـ objects تأخذ الاسماء الإفتراضية مثل CURSOR_1. يمكنك تغيير هذه الاسماء اذا أردت.

احفظ ملف resource، وانسخه فى دليل مؤقت. على القرص المدمج، قد تستخدم ملف الـ Cursor.res فى دليل الـ Data.

٥- يمكنك استخدام الـ Image Editor لإنشاء ملف cursor image، وكذلك أنواع أخرى من الصور مثل bitmaps. ولكن كـ resource، يكون للمؤشر اسم resource والذى يمكن للبرنامج أن يشير إليه- صور المؤشر الباهتة لا تحمل اسماء cursor resource كذلك مرتبط مباشرة داخل الـ exe. الخاص بالبرنامج.

لاستخدام cursor resource، يجب أن يقوم البرنامج بهذه الخطوات الثلاث:

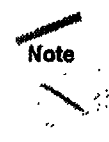
- تجميع الـ resource فى ملف الـ code الخاص بالبرنامج.
- تحميل الـ resource فى الذاكرة أثناء وقت التشغيل.
- حفظ الـ handle الخاصة بالـ resource للاستخدام فى البرنامج.

يهتم Delphi بالخطوة الأولى عندما تقوم بالـ compile والـ link للبرنامج. إنك تؤدى الخطوتين الأخرتين بإضافة تعليمات وعبارات للبرنامج.

٦- ابدأ تطبيقاً جديداً، وفى ملف unit source code، أدخل هذا الأمر بعد الكلمة الرئيسية implementation، بإتباع الأمر المشابه الموجود بالفعل والذى يقرأ ملف dfm. الخاص بالبرنامج. ويجب أن يبدو السطران كما يلى:

```
{$R *.DFM}
{$R Cursor.res}
```

٧- قم بتغيير اسم الملف Cursor.res، اذا لزم الأمر. (يمكنك أيضاً أن تدخل Cursor بدون إمتداد اسم الملف res.). ويخبر الأمر الـ linker بأن يقرأ ملف الـ resource، والذى قد يحتوى على resource متعددة، ويضاف الـ resources فى ملف الـ exe. يمكنك إضافة أى resources من ملف res. باستخدام نفس نوع الأمر، ويمكنك إدخال أى عدد من الأوامر كما تحتاج فى ملف Pascal unit. ولكن فى برمجة Delphi، إنك تقوم بهذا فقط لـ bitmaps، والمؤشرات فى الأيقونات كما هو موضح هنا. وأدوات Delphi الأخرى مثل الـ Menu editor تجعل استخدام الـ resources أمر غير ضرورى.



ملحوظة: إن أمر (اسم ملف $R) يتم تنفيذه أثناء عملية linking و compilation، ولكن ليس فى وقت التشغيل. لا يجب عليك أن توفر ملف الـ res. مع التطبيق التام.

٨- بعد ذلك، أضف الـ code لبرنامج لتحميل صورة المؤشر من ملف exe. الى الذاكرة فى وقت التشغيل. على سبيل المثال، لتحميل وتشغيل المؤشر قبل أن تصبح نافذة الـ form مرئية، قم بإنشاء handler للـ OnCreate event الخاص بالـ form. قم بتعريف ثابت كعدد صحيح لتعريف المؤشر. على سبيل المثال، قم بإدخال هذه الأسطر قبل كلمة البدء الرئيسية:

```
const
  newCursorID = 2;
```

يمكنك استخدام أى قيمة عدد صحيح موجبة لم يتم استخدامها من قبل لمؤشر آخر. اذا كان لديك مؤشرات متعددة. (أو resource أخرى)، قم بتعريف ثوابت أعداد صحيحة لكل منها.

٩- قم بإنهاء OnCreate event handler بإضافة برمجة لتحميل الـcursor resource فى الذاكرة. توضح القائمة (٤-٥) procedure الأخير.

القائمة (٤-٥): تحميل cursor resource

```
procedure TForm1.FormCreate(Sender: TObject);
const
  newCursorID = 2;
begin
  Screen.Cursors[newCursorID] :=
      LoadCursor(HInstance, 'Cursor1');
        Cursor := newCursorID;
end;
```

توضح العبارتان فى القائمة (٤-٥) الطريقة الصحيحة لتحميل cursor resource واستخدامه فى الـ form أو component آخر. ويمثل متغير Screen، وهو من نوع الـ TScreen، يمثل مختلف أحجام الـ display، انظر الـ TScreen فى

online help الخاصة بـ Delphi للحصول على مزيد من المعلومات حول هذا النوع الزائد من البيانات). ويقدم الـ Screen.Cursors قائمة بصور المؤشرات، ومن الناحية الفنية يسمى array property . لتعيين قيمة للـ Cursors، استخدم Windows LoadCursor function كما هو موضح هنا لتحميل cursor images داخل الذاكرة، والذى يسمى هنا 'Cursor1'. (وهذا الـ string لا يعتبر حالة حساسة يمكنك إدخال اسماء resource بحروف صغيرة أو حروف كبيرة). الـ HInstance argument يشير الى الوظيفة الحالية أو instance الخاص للبرنامج.

ويستخدم Windows الـ HInstance. لإيجاد ملف الـ exe. المناسب الذى يحتوى على الـ cursor resource المحدد.

قم بتعيين نتيجة الـ LoadCursor لخاصية الـ Cursors array، وحدد معرف الثابت الواقع بين القوسين. لتحديد المؤشر للـ form، قم بتعيين ثابت الـ Cursor للـ form أو لـ component آخر.

Note

ملحوظة: يحدد الـ Cursor الفردى المؤشر الذى يتم تعينه حالياً، و plural Cursors تشير الى array property التى تعطى كل المؤشرات المتاحة. عندما تقوم بتشغيل البرنامج، قم بتحريك مؤشر الفأرة داخل النافذة لترى شكل المؤشر الجديد.

يمكنك الاستفسار عن صلاحية التعيين للـ Cursors فى القائمة (٤-٥).
بالرغم من المظاهر، ان الـ Cursors لا يعتبر Pascal array، إنما array property. وكذلك، ان الـ Cursors تستدعى فى الحقيقة write procedure والتى تعود وتضيف cursor resources. ان Cursors[X] يبدأ البحث عن مؤشر معرف على أنه X. والتعيين للـ Cursors[X] يحذف أى مؤشر معرف على إنه X قبل إدخال مؤشر جديد فى الخاصية. ومن وراء الستار، يتم التعامل مع الـ Cursors على أنه linked list وهو من الناحية الفنية sparse associated array، ولكن هذه الحقيقة يخفيها الـ TScreen ببراعة.

Tip

فكرة: لا يستطيع Delphi تحميل ملفات res. ذات ١٦ بت - على سبيل المثال، تلك التى هى من تطبيقات الـ Windows 3.1. لتحديثها الى ٣٢ بت، قم بانشاء ملفات resources جديد باستخدام الـ Image Editor.

الرسم بالفأرة:

On the CD-ROM

على القرص المدمج: باستخدام تقنيات الفأرة التى عرفتها فى هذا الباب، يمكنك برمجة عمليات الضغط والسحب لأغراض متعددة. ان التطبيق التالى، Sketch، يوضح الاسس الضرورية للضغط والسحب. ضع هذا المشروع فى دليل الـ Sketch. (يوجد برنامج مشابه فى ادلة الـ Delphi- أستخدم المتوفر فى القرص المدمج المرفق بالكتاب). قم بتشغيل Sketch واضغط واسحب الفأرة داخل نافذة التطبيق لرسم اشكال بسيطة. اضغط مرتين مؤشر الفأرة داخل النافذة هو الـ display توضح القائمة (٤-٦) source code للبرنامج.

القائمة (٤-٦): Sketch\Main.pas

```
unit Main;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics,
Controls, Forms, Dialogs, Menus;

type
  TMainForm = class(TForm)
    MainMenu1: TMainMenu;
      Demo1: TMenuItem;
      Erase1: TMenuItem;
      Exit1: TMenuItem;
      N1: TMenuItem;
      procedure FormMouseDown(Sender: TObject;
    Button:
      TMouseButton; Shift: TShiftState; X, Y: Integer);
      procedure FormMouseMove(Sender: TObject;
    Shift:
      TShiftState; X, Y: Integer);
      procedure FormMouseUp(Sender: TObject; Button:
```

```
        TMouseButton;
        Shift: TShiftState; X, Y: Integer);
        procedure FormDblClick(Sender: TObject);
        procedure Erase1Click(Sender: TObject);
        procedure Exit1Click(Sender: TObject);
        procedure FormCreate(Sender: TObject);
    private
      { Private declarations }
       public
      { Public declarations }
        Dragging: Boolean;
       end;

var
 MainForm: TMainForm;

implementation

{$R *.DFM}

procedure TMainForm.FormCreate(Sender: TObject);
begin
 Dragging := False;
end;

procedure TMainForm.FormMouseDown(Sender: TObject; Button:
 TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
 Dragging := True;
    Canvas.MoveTo(X, Y);
end;
```

```
procedure TMainForm.FormMouseMove(Sender: TObject; Shift:
 TShiftState; X, Y: Integer);
begin
 if Dragging then
    Canvas.LineTo(X, Y);
end;

procedure TMainForm.FormMouseUp(Sender: TObject; Button:
 TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
 Dragging := False;
end;

procedure TMainForm.FormDblClick(Sender: TObject);
begin
 Erase1Click(Sender);
end;

procedure TMainForm.Erase1Click(Sender: TObject);
begin
 Canvas.Brush := Brush; { Assign form's brush to Canvas }
   Canvas.FillRect(MainForm.ClientRect);  { Repaint
  form background }
end;

procedure TMainForm.Exit1Click(Sender: TObject);
begin
 Close;
end;

end.
```

اتبع هذه الخطوات لإعادة إنشاء تطبيق الـ Sketch ولتعرف كيف تنفيذ عمليات الضغط والسحب للفأرة:



١- اذهب الى نافذة الـ unit، وفى TMainForm class، عرف المتغير Dragging، من نوع الـ Boolean. ([انظر القائمة (٤-٦)] يحدد البرنامج هذا المتغير بـ False ليشير الى ان عملية الضغط والسحب قد بدأت. عندما يصبح False، يشير الـ Dragging بذلك الى عمليات الفأرة الطبيعية.

٢- حدد المتغير الـ Dragging بـ False بإضافة OnCreate event الخاص بالـ form. اضف هذه العبارة الى الـ procedure:

```
Dragging := False;
```

٣- لتنفيذ عملية الضغط والسحب، اضف procedure لكل من الـ form events الثلاث، OnMouseDown، OnMouseMove ، وOnMouseUp. أضف البرمجة بعد الخطوة الأخيرة من هذا البرنامج التعليمى.

٤- أضف الـmenu component ، وادخل اوامر لمسح الـ display وانهاء التطبيق. يمكن ان يستخدم أمر المسح عبارات مثل الاتية لملئ الـ client area فى النافذة بلون الخلفية الحالى. أولاً، قم بتعيين خاصية الـBrush للـ form التابعة الـCanvas's Brush، ثم استدع الـ FillRect لملى الـ Canvas بذلك اللون (يشرح الباب ١٣ المزيد حول الـ Canvases والرسوم الجرافيكية):

```
Canvas.Brush := Brush;
Canvas.FillRect(MainForm.ClientRect);
```

٥- خصص Erase procedure الخاص بك الـ OnDblClick التابع للـ form. يمكنك الآن إختيار أمر القائمة أو الضغط مرتين على الفأرة لمسح محتويات النافذة.

٦- اضغط F9 لإجراء عمليتى الـ Compile والـ link وتشغيل التطبيق.

عندما تضغط الفأرة، يحدد OnMouseDown الـDragging flag بـTrue، ويستدعى MoveTo procedure ليماثل موضع الرسوم الجرافيكية إحداثى الفأرة الحالى.

وتأخذ أية رسوم مرئية مكاناً فى هذا الموضع. عندما تطلق الفأرة، يحدد الـ OnMouseUp الـ Dragging بـ False ، مما يلغى عملية الضغط والسحب.

عندما تحرك الفأرة، يقوم الـ OnMouseMove يفحص اذا ما كان الـ Dragging محدد بـ True -اذا كان كذلك، فإنه يستدعى الـ Canvas's LineTo procedure ليرسم خطاً مرئياً من احدث موقع جرافيك حالى الى وضع الفأرة الجديد، والذى يصبح الموقع الجرافيكى لعملية الرسم التالية. اذا كان الـ Dragging محدد بـ False ، لا يقوم الـ OnMouseMove بأداء اية اعمال.

ويعرض تطبيق الـ Sketch الضغط والسحب فقط فهو لا يقوم بالحفاظ على رسومك. بالإضافة الى ذلك، ان تغطية الـ Sketch display بأية نافذة أخرى يؤدى الى مسح الرسوم. يشرح الباب الثلث عشر اساليب إنشاء مزيداً من التطبيقات الجرافيكية العملية.

افكار للمستخدم الخبير

- ان الـ FormKeyDown procedure التابع للـ KeyCount يعين قيمة المفتاح الحالى للـ KeyPressed قبل تشغيل العداد. ان عكس ترتيب هاتين العبارتين سيكون خطأ لان الـ timers تبدأ فى العمل بمجرد ان تحدد خاصية الـ Enabled لهم بـ True. ولا تفعل هذا حتى بعد تحضير أى وكل الظروف المطلوبة من جانب الـ OnTimer event الخاص بـ Timer component.
- يمكنك استخدام الـresource editor مثل الـ Resource Workshop الخاص بـ Borland C++ أو C++ Builder، لإنشاء ملفات res. للإستخدام فى تطبيقات Delphi. قم بضم ملف الـ resource فى ملف الـ code بإضافة الامر {$R اسم ملف res.} يتبع كلمة الـ implementation الرئيسية الخاصة بالـ unit.
- ان واضعى البرامج الخبراء يستخدمون الحروف الكبيرة والصغيرة للمساعدة على التمييز بين تعريفات البرنامج. وطبقاً للعرف يبدأ تعريف الثوابت مثل newCursorId وبأحرف صغيرة، والمتغيرات مثل الـ Count تبدأ بأحرف كبيرة. وتبدأ الـ Procedures مثل LoadCursor ايضاً بأحرف كبيرة وهذه ليست قواعد صارمة، ويمكنك اتباع أسلوب آخر اذا اردت. فقط تأكد من اتباع بعض الأعراف لتجعل الـ code مفهوم نوعاً ما.

● ان برمجة مفتاح الـ Tab تعتبر صعبة فى الغالب بسبب الطريقة التى يستخدم الـ Windows بها هذا المفتاح لتحويل الـ focus من control الى آخر. للتعرف على Tab-key eventsفى تطبيقاتDelphi، إضف البرنامج فى OnKeyUp event . (ان الـ OnKeyDown والـOnKeyPress لا يتلقيان الضغط على مفتاح الـ Tab). على سبيل المثال، إدخل البرمجة من القائمة (٤-٧) فى برامج OnKeyUp event handler الخاص بالـ form لعرض رسالة عندما تضغط الـ Tab أو Shift+Tab. وهذا النص موجود على القرص المدمج فى دليل الـ KeyMouse فى ملف Tab.pas.

القائمة (٤-٧): تلقى الضغط للـ Tab أو Shift+Tab
فى OnKeyUp event handler الخاص بالـ form

```
procedure TForm1.FormKeyUp(Sender: TObject; var Key: Word;
  Shift: TShiftState);
var
  S: string;
begin
  if Key = vk_Tab then
    begin
      S := 'Tab Key Pressed';
        if ssShift in Shift then
          S := 'Shift-' + S;
            ShowMessage(S);
        end;
end;
```

المشروعات التى يمكنك تجربتها

٤-١ : قم بإنشاء نسخة لوحة مفاتيح من برنامج الـ Sketch . إنك تحتاج قليل من قيم الـ Integer لتمثيل موضع الرسم الحالى. قم ببرمجة الـ event handlers الموجهة للزيادة والنقصان فى الوضع الحالى عندما يضغط المستخدم مفاتيح السهام بلوحة المفاتيح، واستدع

CanvasMoveTo والـ LineTo methods التـــابعين للـ form لرسم خطوط .

٤-٢ : اكـتب برنامج اخـتـيـارى يعـرض رسـالة أو صـفـيـراً للـ speaker (الاستجابة بالضبط لا يهم نوعها) عندما يتم ضغط مفتاحى الفأرة . إنك تحتاج الى Boolean flag الذى، عندما يكون True، يشير الى ان زر الفـأرة الأول قـد تم ضـغـطه ولكنه لـم يتـرك الـ event handler للزر الثـانى يمكـن ان يفـحـص مـا اذا كـان الـ flag بـ True، وبهـذه الطريقة، يبرمج الزران على ان يعملا بطريقة مشابهة لمفاتيح الـ Ctrl والـ Alt .

٤-٣ : اضـف custom pen-shaped cursor الى التطبيق الـ Sketch .

٤-٤ : مستوى متقدم . اضف اوامر قائمة للاختيار فيما بين مؤشرات مختلفة لتطبيق الـ Sketch . ملحوظة : قم بتحميل كل cursor resource، تم تعـريفـه بثـابت فـريد، وفى الـ procedures لاوامـر القـائمـة، قـم بتعيين الثابت الذى تم اختياره لخاصية الـ cursor التابعة للـform .

ملخص:

● اضـف keyboard handling للتطبيقات بطريقتين : اسـتخدم الـ Edit أو أى edit component أخـــرى، أو أضـف الـ procedure الـ OnKeyDown والـ OnKeyPress والـ OnKeyUp الخاصة بالـ form .

● يمثل الـ OnKeyDown والـ OnKeyUp مـفـاتيـحـاً تسـتـخـدم الـvirtual key codes مثل الـ vk_up الـ vk_shift . وتمثل الـOnKeyPress مفـاتيح مثل رموز الـ ASCII .

● يسـتـخـدم الـ Windows الـ input focus لتـحـديد اين يرسل الـ events بلوحـــة المفـــاتيح . بشكل طبـيـعى، تذهب مـدخــلات لوحـــة المفـــاتيح الى الـ component الذى يتم اختيارة حالياً . حدد خاصية الـ KeyPreview للـ form بـ True لتعيد توجيه أى نشاط للوحة المفاتيح موجه الى الـ component object الحالى . المحدد لأستقبال أول event من لوحة المفاتيح .

• استـخـدم الـ Timer component لإنشـاء الـ while-key، والذى يفيـد فى أداء الأعمال تكراراً بينما يستمر المستخدم فى ضغط مفتاح بعينه .

• استـخـدم الـ OnKeyPress event لتعـديل قيم المفتـاح . على سبيل المثال، استخدم التقنية الموجودة فى هذا الباب لتغيير مفاتيح الـ Tab لتصبح Enter بحيث يمكن للمستخدمين أن يضغطوا Enter لتحريك input focus من input control الى آخر .

• إن event الفـــأرة OnClick و OnDblClick تعـد كـافيـة لغـالبيـة الأعـمال التى تقـوم بهـا الفـأرة. لمزيـد من التحـكم فى نشاط الفـأرة، أضف الـــ procedure OnMouseDown، والـــ OnMouseMove، والـــ OnMouseUp . وتتلقى هذه الـ events. معلومات إضافية مثل ما اذا كان مفتاحاً مضغوطاً وموقع مؤشر الفأرة.

• استـخـدم الـ Sender parameter، والذى يتم تمريره لأغلب الـ event handlers، لتحديد أى من الـ objects قد تتلقى events . وهذا الأسلوب غاية فى الإفادة عندما يتشارك عدة objects فى نفس الـ event handler .

• قم بتغـيـير خـاصـيـة الـ Cursor لـ component أو form لعـرض شكل cursor مـختلف عندمـا تتـحـرك الفـأرة فـوق الـ form أو الـ component يمكنك الإختيار من بين عدد من المؤشرات المتوفرة .

• يمكنك أيضاً تصميم وتحميل مؤشر محدد كـ resource فى ملف res. . استخدم أمر {اسم ملف $R} لضم بيانات ملف الـ resource الى ملف الـ exe. و event . مثل الـ OnCreate، استدع Windows LoadCursor function وقم بتعيين النتيجة للـ Cursors array الخاصة بالـ component وبالـ form باستخدام ثوابت الأعداد الصحيحة . قم بتعيين الـ index لخاصية الـ Cursor (singular) لتشغيل صورة المؤشر المخصص .

فى البـاب القادم، سوف نشرح واحدة من أهم خصائص واجهة التطبيق للمستخدم فى غالبية تطبيقات الـ Windows : وهو الـ menu bar تعتبر برمجة الـmenu bar أمـر بسـيط مع الـ Menu Designer الخـاص بـ Delphi ويمكنك استخدام عدد من خيارات قوائم برنامجك، مثل الـcheck marks، الأيقونات والقوائم الديناميكية .



الباب الخامس
بنــــاء القــــوائـم

محتويات الباب:

- Components
- Pop-up menus
- Floating pop-up menus
- Dynamic menus
- Form frameworks
- Data modules
- خطوات نحو Menu designer tips
- ActionLists

كما يعرف كل مستخدم لـ Windows ، ان الـ pop-up menu ما هى الا كشف مفصل للأوامر، مثل Open و Exit . فى أغلب التطبيقات، يعطى الـ menu bar اسم واحدة أو أكثر من pop-up menus. يمكنك ايضاً إنشاء floating pop-up menu تظهر عندما تضغط زر الفأرة الأيمن.

ان تصفح قائمة مصممة بعناية يشبه ارتداء معطفك المفضل والانزلاق داخله، ولن تبذل جهداً كبيراً حتى تجعل قوائم تطبيقك مريحة فى الاستخدام كما يوضح هذا الباب، Delphi يقدم ثلاث menu components وعدد من التقنيات المتعلقة بها لبناء قوائم سهلة الاستخدام ذات خصائص معقدة سوف يقدرها المستخدمون.

Components:

ان الـ menu component الثلاثة الخاصة بـ Delphi هى :

- **الـ MainMenu:** أستـخـدم هذا الـ component لإنشـاء الـ menu bar للنافـذة، والذى يتم عـرضـه دائمـاً تحت الـ top border و الـ caption . لإنشـاء قوائم ديناميكية وهى القوائم التى تتغير اثناء وقت التشغيل طبقاً لعمليات البرنامج المتنوعة، مثل فتح نافذة جديدة- يمكنك إضافة MainMenu متعددة على forms وتتبع التعليمات الموجودة فى هذا الباب لدمج اوامر القوائم.
- **PopupMenu:** استخدم هذا الـ component لإنشاء floating pop-up menu، والتى تظهر عندما يضغط المستخدم زر الفأرة الأيمن فوق الـ client area فى النافذة. يمكنك أيضاً تعريف طرق اخرى لعرض floating pop-up menu فى أى موضع من الشاشة.
- **MenuItem:** كل بند فى الـ pop-up menu أو floating pop-up menu يعـد object من الـ TMenuItem class ولكن MenuItem component لا يظهر على VCL palette بالرغم من انه من الاسـهل إنشـاء هـذه الـ objects بالـ Menu Designer الخـاص بـ Delphi، كـمـا يوضح هذا البـاب، يمكنك ايضـاً استخدام عبارات البرنامج أو resource scripts لإنشاء menu items. (بالرغم من ان الـ TMenuItem هو اسم الـ class، الا اننى اشيـر الى هذا الـ component بـ MenuItem من الآن فصاعدا لكى يكون متماشياً مع اسماء الـ components الأخرى).

Pop-Up Menus:

من الناحية الفنية، تعتبر الـ pop-up menu نافذة، والتى تراها عندما تفتح بنداً على menu bar النافـذة. يمكن ان تظهـر floating pop-up menu فى أى مكان على الشاشة. بخلاف. هذا يعتبر نوعان مثل بعضهما. توضح هذا الفصل تقنيـات البـرمـجـة التى تنطبق على كـلا النوعين من pop-up menu اسـتـخـدم MainMenu component لإنشاء pop-up menu على الـ menu bar بالنافذة.
استخدم الـ PopupMenu component لإنشاء floating pop-up menu.

:Main menus

ان معظم برامج الـ Windows لها قائمة رئيسية ، والتى يمكنك إنشاءها باستخدام الـ MainMenu . وكما عرفت من النماذج الأخرى بهذا الكتاب ، فإنك تنشئ قائمة رئيسية بادخال الـ MainMenu على الـ form الرئيسية للبرنامج . اضغط الـ object مرتين لفتح الـ Menu Designer الخاص بـ Delphi، وادخل اوامر برنامجك .

قم بإنشاء قائمة رئيسية للبرنامج باتباع الخطوات الرئيسية الثلاث التالية :

١ - استخدم الـ ProjectlOptions لاختيار Forms page tab وحذف الـ Main للـ form الرئيسية للبرنامج . هذا هو التحديد الافتراضى للتطبيقات ذات النافذة الواحدة .

٢- اضف الـ MainMenu، على الـ form الرئيسية .

٣- حدد خاصية الـ Menu للـ form باسم الـ MainMenu object's .

بالرغم من ان قائمة رئيسية واحدة فقط يمكن ان تكون فى تطبيق ما ، الا من خلال شرطين يمكن لأى form ، حتى الـ dialog box ، ان يكون لها menu bar . ببساطة أضف الـ MainMenu object على الـ form . والشرطان هما : خاصية الـ BorderStyle للـ form قد لا تساوى bsDialog ، والـ FormStyle لها قد لا تساوى MDI child . (قد تمتلك نوافذ الـ fsMDIChild الـ MainMenu object، ولكن كما يوضح الباب الخامس عشر ، تطوير تطبيقات الـ MDI، هذه القوائم قد تم دمجها فى menu bar رئيسية .

: Floating pop-up menus

لإنشاء floating pop-up menu والتى تظهر عندما يضغط المستخدمون زر الفأرة الأيمن، اتبع هذه الخطوات (وكذلك انظر فصل "floating pop-up menu" مؤخراً فى هذا الباب):

١ - أضف الـ PopupMenu1 object على الـ form . يجعل Delphi اسم الـ PopupMenu1 object، حسب النظام الافتراضى .

٢- اضــغــط PopupMenu1 object مــرتين، واســتــخــدم الـ Menu Designer لإدخال اوامر القائمة.

٣- عين الـ popup Menu1 لخاصية الـ PopupMen التابعة للـ form:

: Menu items

يعــتـبـر كل عنصـر فى قـائمــة الـobject من TMenuItem class. يقـوم الـDelphi تلقـائيـاً بإنشـاء هذه الـ objects عندمـا تصـمـم قـوائمك مع الـ Menu Designer. يمكنك رغـم ذلك، إنشـــاء MenuItem objects مع عــبـــارات البـرنامج، وغـالبـاً مـا تشـير الى هذه الـ objects بالعـبـارات. على سـبـيل المثـال، لإضافة علامة صح الى امر القائمة، قم بتعيين True لخاصية الـ Checked الخاصة الـ MenuItem، يمكنك ايضاً أداء عمليات على كل الـ MenuItem المتضمنة فى الـ MainMenuأو الـ PopupMenu بالإشـــــارة الى Items array فى ذلـك الـ object.

كـما هو الحال فى جميع الـ objects باتباع تقليد تسـمية جيـد هو السبيل الى إنشـاء قـوائم قـابلة للحـفظ، يخـتـار Delphi تلقـائيـاً اسـمـاء الـ MenuItems مـثل File1، Open1، و Save2، والتى تعـد مـلائمـة اخير هذه الاسـمـاء مـسـتـخـدمـاً الإرشادات التالية:

- من خلال الـ menu-bar labels، لقـد إضـفـت كلمـة Menu لاسـمـاء ا لـ MenuItem- علـى ســبـــيل المثـــال، FileMenu لـ File menu object، و EditMenu لـ Edit menu.

- للأوامر الخـاصـة فى pop-up menus، لقـد جـعـلت الأوامر الجزء الأول من الـ menu-bar label. على سبيل المثال، MenuItem command objects فى قــائمــة الـ File تســمـى FileOpen، و FileSave، و FileSaveAs. والـ Objects فى قــائمـــة Edit لهــا اســمـــاء مـــثل EditCut و EditCopy، و EditPaste. بعض تطبـيـقـات Delphi تلحق Item الى هـذه الاســمــاء مـثل FileOpenItem، و EditCutItem. قـد تريد ان تفعـل مثلهـا، خـاصـة اذا كنت تخطط لاستخدام الـ wizards لإنشاء application shells.



و Objects مثل FileMenu (التى تمثل قائمة File) و OpenFile (التى تمثل امر الـ Open فى قائمة File) تعد Objects من نفس الـ TMenuItem class، كما هو فى كل بنود القائمة– حتى separator bars التى تقسيم الاوامر الى فئات .

حدد خاصية الـ Caption للـ MenuItem بالنص الذى تريد عرضه فى الـ menu bar أو الأمر . عند تصميم قائمة جديدة، اجد من الاسهل ان اكتب كل الـ Captions الخاصة بقوائمى، ثم اغير الـ Names الافتراضية الخاصة MenuItem objects، ولكن يمكنك أن تدخل هذه القيم بأى ترتيب .

يمكنك ايضاً تعيين قيم لـ Menu item object اثناء التشغيل لإنشاء قوائم ديناميكية تدخل وتخدم الاوامر طبعاً لاحتياجات البرنامج . على سبيل المثال، جرب هذا الخطوات :

١ – أضف الـ MainMenu object على الـ form واضغطها مرتين لفتح الـ Menu Designer .

٢– قم بإنشاء قائمة File أمر Exit (اكتب &File و E&xit لوضع خط تحت مفاتيح الاختصار x و F).

٣– اختر بند كل قائمة إما فى الـ Menu Designer أو فى الـ form، قم بتغيير اسم الخاصية File1 ليصبح FileMenu . غير Exit1 لـ FileExit .

لديك الآن MainMenu يمتلك إثنين من الـ object، الـ MenuItem ، FileMenu و FileExit . يمكنك إدخال عبارات لبرمجة عناصر القوائم هذه لمجموعة من الـ event handler . ثم، أضف هذه العبارة بين البداية والنهاية :

```
with FileExit do
  Visible := not Visible;
```

وكذلك أدخل عبارة ;Close لـ FilelExit command . قم بتشغيل البرنامج و اضغط الزر لتشغيل وإبطال امر الـ Exit . ان استخدام خاصية الـ Visible يعد طريقة فعالة لإنشاء قوائم ديناميكية ذات اوامر تظهر وتختفى طبقاً لظروف البرنامج المختلفة .

أو، يمكنك ابطال بند قائمة بتحديد خاصية الـ Enabled لها بـ True أو False على سبيل المثال، قم بتغيير الـ code السابق الى :



```
with FileExit do
  Enabled := not Enabled;
```

محاكاة الأمر:

لتحاكى اختيار أمراً، إستدع أى MenuItem's Click method. قد يكون هذا مفيداً، مثلاً، عندما تريد أن يقوم الـevent handler بأداء نفس المهمة كأمر pop-up menu. اذا كان command object هذا يعتبر OptionsProject، يمكنك تنفيذ الـevent handler الخاص بالأمر بالعبارة التالية:

```
OptionsProject.Click; {Simulate selecting Options|Project}
```

إن الضغط ليس له تأثير على الـMenuItem object الذى يتصدر قائمة فى الـ window's menu bar رغم ذلك، لا يزال الـMenuItem يتلقى الـOnClick لأى نشاط يقوم به المستخدم مثل ضغط الفأرة على اسم بند قائمة. يمكنك استخدام هذا الـ event لأداء أعمال مثل تغيير اسماء الأوامر فى قائمة، أو إدخال علامات صح.

خاصية الـItems:

يمكنك أيضاً التمكن من الـMenuItem objects بواسطة خاصية الـ Items، والتى تعتبر عنصراً فى الـMainMenu، والـPopupMenu، والـ MenuItem. وتعتبر الـ Items object من الـTMenuItem التى يمكنك استخدامها لإستدعاء Methods مثل الـItems.Insert، أو الـarray فى تعبيرات مثل MyMenu.Items[N]. فى الـMainMenu object، تحتوى خاصية الـ Items على عناصر pop-up menu الخاصة بالـmenu bar، على سبيل المثال، فى برنامج ذى ثلاث pop-up menus-File، و Edit، و Help- تقوم العبارات التالية بتغيير اسم القائمة الأولى الى Presto-Chango، وتبطل قائمة الـEdit، وتجعل قائمة Help غير مرئية:

```
with MainMenu1 do
begin
    Items[0].Caption := 'Presto-Chango';
    Items[1].Enabled := False;
    Items[2].Visible := False;
end;
```



لكل MenuItem object أيضاً خاصية Items، التى يمكنك استخدامها للتوصل الى أوامر القائمة، على سبيل المثال، لإبطال أمر الـ File|Exit، بفرض أن هذا هو الأمر الوحيد فى القائمة، يمكنك استخدام هذه العبارة:

```
FileMenu.Items[0].Enabled := False;
```

وهذا له نفس التأثير الذى تحدثه العبارة التالية، والتى هى أبسط فى أغلب الحالات:

```
FileExit.Enabled := False;
```

وكقاعدة عامة، لأداء مهام الـ MenuItem objects، من الأيسر أن تشير الى اسماء الـ object، كما هو الحال فى المثال السابق. لتحديد خصائص متعددة رغم ذلك، قد يكون من الأفضل أن تستخدم خاصية الـ Items. أى من الأسلوبين يتوصل الى نفس MenuItem objects، لذا استخدم ما يحلو لك. على سبيل المثال، العبارة التالية تستخدم متغير Integer فى for Loop لإبطال كل الأوامر فى الـ FileMenu:

```
for I := 0 to FileMenu.Count - 1 do
  FileMenu.Items[I].Enabled := False;
```

وبالمناسبة، لأن العبارة السابقة تشير الى الـ FileMenu مرتين، يمكنك تبسيط الـ code باستخدام عبارة with. رغم أن النتيجة تحمل سطراً إضافياً، ولكنها أكثر فعالية من الناحية المنطقية:

```
with FileMenu do
  for I := 0 to Count - 1 do
   Items[I].Enabled := False;
```

وعبارة الـ with تخبر Delphi أن يستخدم الـ Count والـ Items التابعين للـ FileMenu، مما لا يدفع الحاجة الى أن تسبق هذه الاسماء FileMenu ونقطه. وهذا يوفر الكتابة، ويمكن أن يساعد الـ compiler على إنتاج code أكثر فعالية. حاول استخدام with قدر استطاعتك.

وخاصية الـ Count للـ FileMenu تساوى عدد الـ MenuItem objects فى الـ Items array. ولأن array index الأول يساوى صفر، فإن object

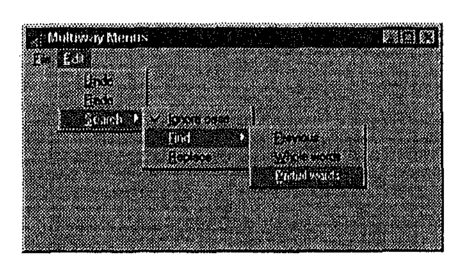
index الأخير يساوى Count - 1. ضع هذا فى الإعتبار دائماً عند كتابة loop للتعامل مع خاصية الـ Items كـ Items.

:Multiway menus

يمكن لأى بند فى الـ pop-up menu أن يكون له قائمة فرعية متصلة به. كما يوضح الشكل (٥-١)، يمكنك أن تحمل هذه الفكرة الى أقصى مدى، ولكن الأفضل ألا تذهب بعيداً. بالرغم من أن multiway menus تعتبر نافعة لتنظيم هياكل الأوامر المعقدة، ولكن يمكن أن تكون صعبة الاستخدام.

Note

ملحوظة: يقوم Windows 95 بتنشيط إختيار القائمة بألا يطالب المستخدمين بضغط زر الفأرة عدة مرات. وهذا يجعل multiway menus أسهل فى الاستخدام أكثر منها فى النسخ السابقة من الـ Windows. إن الإكتفاء بتكوين مستويان يعد من أفضل الأمور. ،تعتبر الثلاث مستويات دائماً عدد كبير جداً.



شكل (٥-١): Multiway menus توضح تداخل هياكل الأوامر المعقدة. هذا العرض من برنامج الـ MultiMen على القرص المدمج المرفق بالكتاب

لإنشاء قوائم فرعية فى الـ Menu Designer، اتبع الخطوات التالية:

١- قم بإنشاء أمر قائمة كـ search [أنظر الشكل (٥-١)].

٢- قم بإبراز الأمر واضغط السهم الأيمن Ctrl+right. هذا يؤدى الى إنشاء قائمة فرعية يمكنك التعامل معها كما تفعل فى أى من pop-up menu windows أخرى.

٣- أعد هذه الخطوات لإنشاء مستويات تداخل إضافية.

ولا تتطلب الـMultiway menus برمجة خاصة. فإنك تنشئ event handlers لأوامرها بنفس الطريقة التى تفعلها فى pop-up menus ذات المستوى الواحد. بعبارة أخرى، تداخل القوائم أمر بصرى خالص. لذلك، بصرف النظر عن مستوى التداخل، يجب أن يكون لكل MenuItem objects فى الـ MainMenu أو الـ PopupMenu خصائص Name فريدة.

:Menu item shortcut keys

يمكنك أن تصنع نوعين من الـ shortcut key يتم تحديدهما لأوامر القائمة. يمكنك أن تسبق الحرف بـ & لتصمم مفتاح Alt. على سبيل المثال، اذا حددت الـ Caption الخاص بالـ MenuItem بـ &Configure، يمكن للمستخدمين أن يضغطوا Alt+C لفتح قائمة الـ Configure.

والنوع الثانى من shortcut key يسمى accelerator. من المتعارف عليه أن accelerator- عبارة عن مجموعة من مفاتيح عددية أبجدية، Ctrl، Shift، سهم، و function keys وتسمى accelerators. لأن المستخدم يستطيع أن يضغطها لإختيار أوامر دون الحاجة لفتح نافذة قائمة.

لاتقم بإضافة الـshortcut key labels فى الـ MenuItem Captions. يقوم Delphi تلقائياً بإلحاق اسماء المفاتيح. على سبيل المثال، إدخل E&xit للـ Caption الخاص بـ FileExit، وحدد shortcut key هذا الـ object بـ Alt+X. يقوم Delphi تلقائياً يعرض الأمر على انه Exit Alt+X فى pop-up menu.

يمكنك اختيار shortcut keys من قائمة الائحة التابعة للخاصية. ولكن، لان هذه القائمة ليس لها مداخل عديدة، اجد انه من الايسرْ ان تدخل key label كنص. فمثلاً، بدلاً من ان تضغط A L T + X، إننى اكتب الخمسة رموز A L T + X فى خاصية الـ ShortCut واضغط Enter. يجب ان تفعل هذا لتعيين مفاتيح مثل Enter والتى لا توجد فى قائمة اللائحة- اكتب E N T E R، مثلاً. يمكنك ايضاً إنشاء إختصارات رموز بادخال A, B, C, 1, 9 أو أى مفتاح آخر تستطيع كتابته.

ويضع الجدول (٥-١) كشفاً بالـ standard accelerator shortcut key لبنود القائمة العامة. وتتبع تطبيقات الـ Windows 95 هذه المعايير أكثر منها فى نسخ الـ Windows السابقة، وإنها فكرة جيدة يجب ان تستخدمها فى تطبيقاتك.

وهذه ليست قواعد جامدة، ان استخدام المفتاح يعتمد على التطبيق- فمثلاً، يمكن ان تستخدم مفتاح Ins لإدخال حقل بدلاً من ربط زر insert/overtype الخاص بالـ editor. (إننى لا إتفق على ان الـ FilelExit هو المعيار الرسمى لمفتاح الـ Alt+F4. فالجميع يعرفون ان Alt+X هو accelerator standar للخروج من البرنامج).

جدول (٥-١): Standard Accelerator Shortcut Keys

Command	Menu	Shortcut
Cascade	Window	Shift+F5
Collapse all	Tree	Ctrl+* (keypad)
Collapse item	Tree	- (keypad minus)
Copy	Edit	Ctrl+C
Cut	Edit	Ctrl+X
Delete	Edit	Del
Exit	File	Alt+X (officially Alt+F4)
Expand all	Tree	* (keypad star)
Expand item	Tree	+ (keypad plus)
Expand/Collapse	Tree	Enter
Find	Edit	Ctrl+F
Find next	Edit	F3
Insert/Overtype	Edit	Ins
New	File	Ctrl+N
Open	File	Ctrl+O
Paste	Edit	Ctrl+V
Print	File	Ctrl+P
Replace	Edit	Ctrl+H
Save	File	Ctrl+S
Search	Help	F1
Select all	Edit	Ctrl+A
Tile	Window	Shift+F4
Undo	Edit	Ctrl+Z

علامات الصح:

إن إضافة وإزالة علامات الصح من إوامر القائمة لن يكون سهلاً. ببساطة، حدد أى خاصية Checked للـ MenuItem بـ True لعرض علامة صح، أو بـ

False لمسح إحدى العلامات . يمكنك القيام بهذا أثناء وقت التصميم أو وقت التشغيل باستخدام عبارة مثل :

```
EditInsert.Checked := True;
```

فى تصميم واجهة التطبيق للمستخدم الخاص بالبرنامج ، فكر جيداً اذا كنت سوف تجعل أمر ذو علامة صح يعمل كـ toggle أو كإختيار من مجموعة من الأوامر المرتبطة بها . على سبيل المثال ، قد تعطى قائمة الـ Level الخاصة بالألعاب مستويات صعوبة ، واحداً فقط هو الذى يمكن إختياره فى كل مرة . وأمر الـ Insert فى قائمة الـ Edit من المحتمل أن يكون للـ toggle والحل .

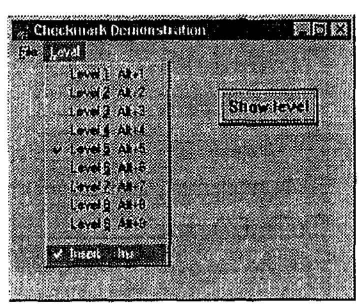
قم بإنشاء toggle باستخدام عامل not للـ Pascal كما فى هذه العبارة ، والتى قد تدخلها فى OnClick event handler الخاص بالـ MenuItem .

```
with EditInsert do
  Checked := not Checked;
```

فى أى مكان آخر فى البرنامج ، استخدم الـ Checked فى عبارة if لإختيار عمل :

```
if EditInsert.Checked then
  ... { do something if menu item is checked }
```

قم بتشغيل الـ Checks وجرب قائمة الـ Level . ويستخدم البرنامج علامات صح بطريقتين : أن تختار من بين تسعة مستويات ، ليجعل أمر Insert من الـ on الى off اضغط زر النافذة لعرض مواصفات المستوى الحالى . يوضح شكل (٥-٢) نافذة البرنامج .



شكل (٥-٢): تعرض تطبيقات الـ Checks كيفية استخدام علامات الصح كـ toggle وتختار واحداً من مجموعة من الأوامر، مثل المستويات التسع الموضحة هنا

توضح القائمة (٥-١) الـform's source code والتى تظهر كيف يمكن أن تتـشـارك سلسلة من أوامـر القـائمـة فى نفس الـevent handler لإنشـاء الـ LevelClick، عرفه كما هو موضح فى TMainForm class، وأدخل عبارات الـ procedure فى الـ implementation section يقـوم الـ procedure بإغـلاق علامات الصح للأوامر من Level1 الى Level9، ثم يحدد خاصية الـ Checked للـ Sender بـ True. إن تعبير الـ TMenuItem(Sender) تجعل الـ Sender كـ TMenuItem object. لاستخدام الـLevelClick، قم بتعيينه للـ OnClick event لكل MenuItem، من الـ Level1 الى Level9.

القائمة (٥-١):Checks\Main.pas

```
unit Main;

interface

uses
 Windows, SysUtils, Messages, Classes, Graphics, Controls,
    Forms, Dialogs, Menus, StdCtrls;

type
 TMainForm = class(TForm)
      MainMenu1: TMainMenu;
         FileMenu: TMenuItem;
         FileExit: TMenuItem;
         LevelMenu: TMenuItem;
         Level1: TMenuItem;
         Level2: TMenuItem;
         Level3: TMenuItem;
         Level4: TMenuItem;
         Level5: TMenuItem;
         Level6: TMenuItem;
         Level7: TMenuItem;
         Level8: TMenuItem;
         Level9: TMenuItem;
```

```
        ShowButton: TButton;
          N1: TMenuItem;
          LevelInsert: TMenuItem;
          procedure FileExitClick(Sender: TObject);
          procedure LevelClick(Sender: TObject);
          procedure ShowButtonClick(Sender: TObject);
          procedure LevelInsertClick(Sender: TObject);
  private
          { Private declarations }
    function GetLevel: Integer;
        public
       { Public declarations }
        end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}

const
  highLevel = 9;   { Highest level command }

procedure TMainForm.FileExitClick(Sender: TObject);
begin
  Close;
end;

procedure TMainForm.LevelClick(Sender: TObject);
var
  I: Integer;
begin
```

```
    with LevelMenu do
          for I := 0 to highLevel - 1 do
          items [I]. checked := fales;
               .. TMenuItem(Sender).Checked := True;
end;

function TMainForm.GetLevel: Integer;
var
  I: Integer;
begin
  GetLevel := -1;
     with LevelMenu do
       for I := 0 to highLevel - 1 do
               if Items[I].Checked then
       GetLevel := I + 1;
end;

procedure TMainForm.ShowButtonClick(Sender: TObject);
var
  S: string;
begin
  S := 'Level = ' + IntToStr(GetLevel) + ' - Insert:';
     if LevelInsert.Checked
       then S := S + 'ON'
          else S := S + 'OFF';
         ShowMessage(S);
end;

procedure TMainForm.LevelInsertClick(Sender: TObject);
begin
  with Sender as TMenuItem do
       Checked := not Checked;
end;
```

end.

وتوضح Function GetLevel كيفية فحص الـ MenuItem object من خلال Items array لتحديد أى من الاوامر له علامة صح. (-1) function اذا لم يكن هناك أمراً مشار إليه بعلامة صح، أو انه يرجع المستوى من ١ الى ٩ .

عندما تضغط الزر، يقوم الـ ShowButtonClick بإنشاء string يوضح المستوى الحالى (باستدعاء الـ GetLevel function) والتحديد الحالى للـ Toggle Insert . والـ ShowMessage procedure الخاص بـ Delphi يعرض النتيجة النهائية .

: Floating Pop-up Menus

يوجد ثلاث إختلافات هامة فقط بين الـ floating والـ nonfloating pop-up menus وهى المكان، المكان، المكان . إنك تقوم ببرمجة الـ PopupMenu objects بنفس الطريقة التى تبرمج بها الـ MainMenu، ولكن بدلاً من الظهور فى الـ menu bar، يمكن ان تظهر floating pop-up menu فى أى مكان من الشاشة . وتطبيق الـ MemoPad، من الباب الثانى، يظهر كيفية انشاء واستخدام الـ floating pop-up menu.

لإنشاء الـ floating pop-up menu، أضف PopupMenu component على الـ form، اضغط الـ object مرتين، إدخل الأوامر باستخدام الـ Menu Designer بنفس الطريقة التى تتبعها مع الـ MainMenu. قم بتعيين الـ PopupMenu لخاصية الـ PopupMenu بالـ form .

فى الغالب تتشارك اوامر floating pop-up menu و menufloating pop-up فى نفس event handlers كما فى القائمة الرئيسية للبرنامج، ولكن هذه ليست قاعدة . يمكنك برمجة الـ event handlers بشكل منفصل للـ floating pop-up menu اذا اردت .

زر الفأرة الأيمن:

لبرمجة الضغط لزر الفأرة الأيمن لعرض الـ floating pop-up menu، قم بتعيين اسم الـ PopupMenu لخاصية الـ PopupMenu بالـ form . تظهر القائمة عندما يضغط المستخدمون زر الفأرة الأيمن فى نافذة الـ form .

يمكنك ايضاً تعيين الـ PopupMenu فى وقت التشغيل ، والذى قد تفعله للتغيير بين اثنين أو أكثر من الـ floating pop-up menu إعتماداً على الظروف المختلفة للبرنامج فى event handler للزر ، أو فى الـprocedure لأمر القائمة ، استخدم عبارة مثل التالية لتعيين الـ PopupMenu لخاصية الـ PopupMenu التابعة للـ form :

```
PopupMenu := PopupMenu2;  { Assign PopupMenu2 to
PopupMenu }
```

طرق أخرى لـ float:

استدع الـ Popup method التابع لـ PopupMenu لعرض الـ floating pop-up menu إستجابة لـ events مختلفة عن ضغط زر الفأرة الأيمن قم بتمرير احداثيات الشاشة x و y للـ Popup كما فى هذه العبارة :

```
PopupMenu1.Popup(100, 100);
```

إحداثيات الـx و الـ y الخاصة بالـ Popup التى لها علامة بشاشة الـ Windows ، ويكون (٠ ، ٠) الركن الايسر العلوى للشاشة .

عند تمرير احداثيات ضغط الفأرة للـ Popup ، لان هذه القيم لها علاقة بالنافذة ، استدع ClientToScreen function للتحويل الى الاحداثيات ذات العلاقة بالشاشة . أولاً قم بتعريف متغير من نوع الـ TPoint

```
var
  Pt: TPoint;
```

ثم ، فى MouseDown التابع للـform ، ادخل هذه العبارات :

```
Pt.X := X;
Pt.Y := Y;
Pt := ClientToScreen(Pt);
PopupMenu1.Popup(Pt.X, Pt.Y);
```

يعين أول سطرين إحداثيات الفأرة x و y اللذان تم تمريرهما الـ event handler الى متغير الـ TPoint . يستدع الخط الثالث الـClientToScreen فى TControl class ، والتى تعتبر واحدة من السلالات البعيدة للـ TForm . وهذا

يحول الـ Pt من الاحداثيات المتصلة بالـ client-relative من النافذة الى القيم المتصلة بالشاشة . ويقوم السطر الأخير لتمرير قيم الـx و الـ y المحولة لـ method الـ Popup التابعة لـ PopupMenu .

Dynamic Menus:

يقدم Delphi تقنيات متنوعة لإنشاء قوائم تتغير بطريقة ديناميكية فى وقت التشغيل . يمكنك ان :

- تغير menu bar كامل للنافذة .
- تدخل وتحذف pop-up menu .
- تضيف وتزيل الأوامر .
- تدمج الـ menu objects .

توضح الفصول التالية هذه التقنيات وما يزيد .

تغيير القوائم:

ان اسهل طريق لتغيير قائمة النافذة لتعيين MainMenu مختلف لخاصية Menu التابعة للـ form أولاً ، إدخل إثنين أو أكثر من الـ MainMenu الـ form واستخدم الـ event handler لإدخال اوامره . ثم ، ربما فى برنامج Menu Designer لزر أو لأمر ، ادخل عبارة مثل :

```
Menu := MainMenu2;
```

عندما تقوم بتشغيل البرنامج ، يتغير الـmenu bar من التعيين الخاص به فى وقت التصميم ، الى MainMenu2 . يقوم للـ menu bar تلقائياً بتغيير مظهره البصرى عندما ينفذ البرنامج عبارة التعيين السابقة .

ادخال وحذف القوائم:

يمكنك دمج الـ menu object (انظر دمج MainMenu objects ' ، مؤخراً فى هذا الباب) ولكن هناك اسلوباً ابسط يمكنك استخدامه لادخال وحذف pop-up menus بأكملها فى menu bar بالنافذة . لتجربة هذا الاسلوب ، أضفت MainMenu object منفرداً على الـ form إضغط الـ object مرتين ، واستخدم الـMenu Designer لإدخال قوائم وأوامر .

بعد ذلك، قد بإنشاء الـ OnCreate الخاص بالـform ادخل عبارات مثل التالية لجعل القوائم القافزة غير مرئية:

```
OptionMenu.Visible := False;
WindowMenu.Visible := False;
```

أخيراً، استجابة لضغط زر ما أو أمر قائمة (أو أى حدث آخر)، حدد خاصية الـ Visible لعنصر قائمة بـ True. تظهر القائمة تلقائياً. وهذا يعد اسلوباً جيداً لبرمجة امر الـ Advanced Menus السريع الذى يمد القوائم الى أقصى مدى لها. وتبقى بنود القوائم الغير مرئية نشطة، ولاتزال تتلقى الـOnClick events. اذا كان العناصر القوائم هذه shortcut key مرتبطة بها، فيمكن للمستخدمين اختيارها حتى اذا كانت الأوامر غير مرئية.

تغيير عناصر القائمة:

لتغيير نص أمر ما، ببساطة قم بتغيير Caption جديد لأى MenuItem object. على سبيل المثال، يمكنك برمجة امر الـ Undo لإختار المستخدمين بما يمكن إبطاله:

```
EditUndo.Caption := '&Undo deletion';
```

إضافة، إدخال، وحذف عناصر القائمة:

يمكنك إضافة، وإدخال، وحذف اوامر من أى pop-up menu من TMenuItem class. على سبيل المثال، يمكنك إلحاق اسماء ملفات الى نهاية-.FileMenu object بقائمة File. أو، يمكنك استخدام هذا الاسلوب لتعديل اوامر pop-up menu فى وقت التشغيل.

والخطوة الأولى فى إضافة أو إدخال بند قائمة هى ان تنشئ الـTMenuItem object. أولاً، قم بتعريف متغير مثل هذا الذى فى الـevent handler:

```
var
 MI: TMenuItem;
begin
 ...
end;
```

فى كيـان الـ procedure ، قم بإنشـاء TMenuItem جـديد باستـدعـاء الـ Create method ، وقم بتـمـرير الـ object الاساسى ، والـذى يكون فى الغـالب القائمة الأمر . لبند الأمر الجديد :

```
MI := TMenuItem.Create(FileMenu);
```

لديك الآن menu object جديد يُشار إليه بـ MI . قم بتعيين الـ caption الخاص بالـ object's والخصائص الأخرى له باستخدام عبارات مثل التالية :

```
MI.Caption := '&New command';
MI.Visible := True;
MI.ShortCut := ShortCut(vk_F1);
```

عندما تنتهى من تشكيل عنصر القائمة الجديد ، أضـفة الى اسفل pop-up menu باستدعاء Add :

```
FileMenu.Add(MI);  { Add menu item to bottom of File
menu }
```

عند هذه النقطة ، يفترض FileMenu object أن يحتوى الـ MI ، ويحذف هذا object فى الوقت المناسب . لا يجب عليك حذف الذاكرة المخصصة للـ MI من جانب الـ Create .

يمكنك استخدام MI مرة اخرى لإنشاء عنصر قائمة آخر ، ولكن لكل مرة ، يجب عليك إستدعاء الـ Create method كما هو موضح هنا . على سبيل المثال ، لا يمكنك تعيين caption مختلف للـ MI وإضافته للقائمة . لواضعى برامج Pascal الخبراء : يعتبر الـ MI مشيراً الى object تم تخصيصه بطريقة ديناميكية . فى البرامج ، لا يجب عليك إبطال مرجعية هذا الـ Pointer - ببساطة استخدمه كما كنت تستخدم متغير من نفس النوع .

أو ، يمكنك إضافة بند قائمة بين عناصر اخرى . افترض ان لديك خط فاصل يسمى FileSep . لإدخال MI فوق هذا الخط ، أولاً حدد موضع فهرس الفاصل وقم بتعينه لمتغير Integer :

```
Index := FileMenu.IndexOf(FileSep);
```

ثم ، بدلاً من استدعاء Add ، ادخل MI باستدعاء الـ Insert method وقم بتمريرها على قيمة الـ Index وشكل القائمة الجديد :

```
FileMenu.Insert(Index, MI);  { Insert menu item at Index }
```

يمكنك ايضاً حذف عناصر القائمة بواسطة قيمة الفهرس . استدع Delete الـmethod الخاص بالـ TMenuItem :

```
FileMenu.Delete(Index);
```

أو، يمكنك حذف بند قائمة باستدعاء Remove . على سبيل المثال، هذا يحذف أمر الـ Exit من قائمة الـ File :

```
FileMenu.Remove(FileExit);
```

يمكنك تعديل محتويات قائمة رئيسية أو pop-up menu من خلال خاصية الـ Items . استخدم الـ Items كـ array من TMenuItem object . على سبيل المثال، MyMenu.Items[I] تشير الى الـ TMenuItems للـ index [I] في MyMenu .

تعيين code للـ menu item:

عند إنشاء عنصر قائمة جديد، يجب عليك ايضاً تعيينه الى event handler لأداء عمل عندما يختار المستخدمون الأمر . وهذا يأخذ ثلاث خطوات :

١- قم بتعريف الـ event handler procedure و الـ form's class .

٢- قم بتنفيذ الـ procedure في implementation section .

٣- قم بتعيين الـ procedure لـ OnClick الخاص بعنصر القائمة .

على سبيل المثال، اضف التعريف التالي TForm1 class (في الغالب أضف هذا التعريف وأي تعريف أخر في قطاع الـ public أوالـ private للـ class في هذه الحالة، ضعه في الـprivate) :

```
procedure NewCommandClick(Sender: TObject);
```

بعد ذلك، قم بتنفيذ الـ procedure في أي مكان بقطاع الـ implementation :

```
procedure TForm1.NewCommandClick(Sender: TObject);
begin
  ShowMessage('New command executed!');
end;
```

لاحظ ان تعريف الـ procedure كـ method فى قطاع implementation باستخدام الـ class name ، ونقطة ، واسم الـ procedure بهذه الطريقة ، ليصبح الـ method تابعاً للـ form object الذى تم إنشاءه باستخدام الـ TForm1 class .

أخيراً ، قم بتعريف متغير TMenuItem مثل MI ، قم بإنشاء الـ object ، عين الخصائص التى تريدها ، وعين الـ event handler للـ OnClick event :

```
MI := TMenuItem.Create(FileMenu);
MI.Caption := '&New command';
MI.OnClick := NewCommandClick;
```

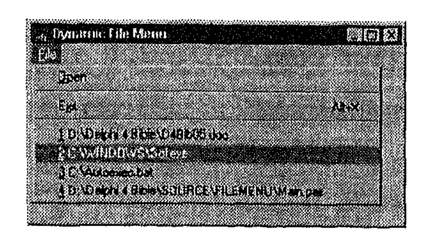
ادخل أو أضف الـ object الجديد فى القائمة . عندما يختاره المستخدمين ، يستدع البرنامج الـ NewCommandClick .

بالطبع ، يمكنك تعيين الـ event handler لأمر قائمة موجود بالفعل الى OnClick event. الخاص بند قائمة جديد . فقط استخدم السطر الأخير من المثال السابق ، وقم بتغيير .NewCommandClick الى اسم الـ procedure الموجود بالفعل .

إضافة اسماء ملف لقائمة ملف:

لا يعتبر الـ MenuItem objects أمراً . فيمكن ان يكونوا لإعطاء المعلومات ايضاً-على سبيل المثال ، اسماء الملفات التى تم فتحها مؤخراً . يمكنك استخدام الـ Add ، Insert ، و Delete للتحكم فى هذا الكشف - فقط قم بإنشاء الـ TMenuItem objects وعين اسماء ملف لخصائص الـ Caption . رغم ذلك ، إننى اجد من الايسر إدخال اوامر فى قائمة File وتعيين اسماء ملف لها بمجرد ان يفتح المستخدمون ملفات جديدة .

On the CD-ROM **على القرص المدمج:** ان تطبيق الـ FileMenu الموجود على القرص المدمج المرفق بهذا الكتاب فى دليل الـ Source\FileMenu يوضح الـ code الضرورية . يوضح شكل (٥-٣) قائمة الـ File للبرنامج مع أربعة ملفات تم فتحها حدثياً . يمكنك اختيار تلك الاسماء مثل أى أمر آخر ، ولكل منها مفتاح Alt+N ، حيث ان الـ N هو الرقم الذى تحته خطاً . ان الـ code التى تتحكم فى كشف اسم الملف تعتبر محيرة الى حد ما ، كما توضح القائمة (٥-٢) .



شكل (٥-٢): تقنية واجهة تطبيق عامة تضع كشفاً بالملفات التى تم فتحها حديثاً

فى قائمة الـFile. قد تكون البرمجة محيرة، كما يوضح تطبيق الـ File menu.

القائمة (٥-٢):T Filemenu\Main.pas

```
unit Main;

interface

uses
  Windows, SysUtils, Messages, Classes, Graphics, Controls,
    Forms, Dialogs, StdCtrls, Menus, Buttons;

type
  TMainForm = class(TForm)
      MainMenu1: TMainMenu;
        FileMenu: TMenuItem;
        FileExit: TMenuItem;
        OpenButton: TButton;
        OpenDialog: TOpenDialog;
        FileOpen: TMenuItem;
        FileSep1: TMenuItem;
        FileSep2: TMenuItem;
        FileName1: TMenuItem;
        FileName2: TMenuItem;
        FileName3: TMenuItem;
        FileName4: TMenuItem;
```

```
        BitBtn1: TBitBtn;
          procedure FileExitClick(Sender: TObject);
          procedure OpenButtonClick(Sender: TObject);
          procedure FileName1Click(Sender: TObject);
         private
        { Private declarations }
         public
        { Public declarations }
         end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}

procedure TMainForm.FileExitClick(Sender: TObject);
begin
 Close;
end;

{- Prompt for filename and add name to File menu }
procedure TMainForm.OpenButtonClick(Sender: TObject);
var
 S: string;
   I, K: Integer;
begin
 if OpenDialog.Execute then with FileMenu do
    begin
     if not FileSep2.Visible then
          FileSep2.Visible := True;  { Make separator visible }
           K := IndexOf(FileName1);
```

```
      for I := Count - 1 downto K + 1 do
        begin  { Move current filenames down one position
      }
          S := Items[I - 1].Caption;
            S[2] := Chr(Ord('0') + (I - K + 1));  {
        Alt-Shortcut }
            Items[I].Caption := S;
            Items[I].Visible := Items[I - 1].Visible;
          end;
        FileName1.Caption := '&1 ' +
      OpenDialog.Filename;
        FileName1.Visible := True;
        ShowMessage('Adding: ' + OpenDialog.Filename);
       end;
end;

{- Get filename selected from File menu }
procedure TMainForm.FileName1Click(Sender: TObject);
var
 Filename: string;
begin
 with Sender as TMenuItem do
    begin
     Filename := Caption;
       System.Delete(Filename, 1, 2);
      end;
    ShowMessage('Selected: ' + Filename);
end;

end.
```

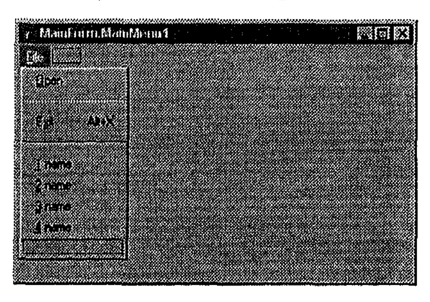
Tip

فكرة: يوضح الـ FileName1Click فى القائمة (٥-٢). كيفية حل التداخل الذى قد يسبب لك مشكلة من وقت إلى آخر. ان الـ procedure يستدعى Pascal Delete procedure لإزالة Alt key

labels التى تحتها خطاً المضافة لكل اسم ملف . رغم ذلك، تخبر عبارة with السابقة Delphi ان يستخدم اعضاء الـTMenuItem ، واحداً منها يدعى Delete ايضاً. لقد وجدت حلاً لتداخل الاسماء هذا بأن سبقت Delete بـ System ونقطة، والتى تقول للـ compiler ان يستخدم Delete string procedure الموجود فى الـ System unit، وليس TMenuItem method الذى يحمل نفس الاسم. والاسماء المشتركة الواسعة الاستخدام الأخرى والتى قد تحتاج إلى تأهيلها تتضمن Insert، Text ، Assign ، و Close .

اتبع هذه الخطوات لإنشاء قائمة File وإضافة اسماء الملفات التى تم فتحها حديثاً فى اسفل نافذة القائمة :

١ - أضف Menu component على الـ form واستخدم الـ Menu Designer لإنشاء قائمة File ذات فاصل (أضف هذه العلامة (-) فى خاصية الـCaption) واربعة اسماء ملف وهمية لعناصر القائمة. انظر شكل (٥-٤) كمثال. إجعل اسم القائمة FileMenu، واسماء الملفات الوهمية FileName1، FileName2، FileName3، FileName4 . إجعل اسماء الفواصل FileSep1، و FileSep2 . جدول (٥-٢) يعطى اسماء وخصائص الـ component الهامة .



شكل (٥-٤): قائمة تطبيق الـ FileMenu التى تم فتحها فى الـMenu Designer الخاص بـDelphi، مع اربعة عناصر اسماء ملفات وهمية

٢- حدد الـ Visible بـ False للفاصل وكل بنود القائمة الوهمية. ان البرنامج يجعلها مرئية عندما يفتح المستخدمون ملفات جديدة.

٣- قم بإنشاء الـ event handler للمدخل الوهمى الأول باختياره فى الـ Menu Designer . انظر FileName1Click فى القائمة، والتى تعرض اسم الملف المختار. عين نفس هذا الـ event handler إلى البنود الوهمية الأخرى.

٤- إضف الـ OpenDialog object على الـ form.

٥- قم بإنشاء الـ event handler لأمر الـ Open أو الزر ما، وأضف البرمجة من الـ OpenButtonClick procedure.

استخدم shortcut keys:

ان Menus unit الخاصة بـ Delphi، مضافة إلى أى برنامج مع MainMenu objects أو PopupMenu، تقدم اربعة نظم فرعية تفيد فى تعيين استخدام accelerator shortcut keys. للحصول عليها، أضف MainMenu object على الـ form، واستخدم الـ Menu Designer لإنشاء قائمة ذات عنصراً واحداً على الأقل يسمى Demo. إننى استخدم الاسم الافتراضى، هنا وهو Demo1.

جدول (٥-٢): خصائص الـ FileMenu Application Component

Component	Name	Property	Value
Form	MainForm	Caption	Dynamic File Menu
Button	OpenButton	Caption	Open file
BitBtn	BitBtn1	Kind	bkClose
OpenDialog	OpenDialog	Filter	All files (*.*)|*.*
MainMenu	MainMenu1		
MenuItem	FileOpen	Caption	&Open
MenuItem	FileExit	Caption	E&xit
MenuItem	FileSep1	Caption	-
Component	Name	Property	Value
MenuItem	FileSep2	Caption	-
		Visible	False
MenuItem	FileName1	Caption	&1 name
		Visible	False
MenuItem	FileName2	Caption	&2 name
		Visible	False
MenuItem	FileName3	Caption	&3 name
		Visible	False
MenuItem	FileName4	Caption	&4 name
		Visible	False

استخدم الـ ShortCut function لتعيين accelerator key لعنصر قائمة.

يعرف Delphi الـ function كما يلى:

```
function ShortCut(Key: Word; Shift: TShiftState): TShortCut;
```

باستخدام ShortCut، يمكنك تعيين F9 لخاصية الـ ShortCut key الخاص بـDemo1 (للاستمرار، اضف هذه العبارة لبرنامج الـ event handler الخاص بالزر):

```
Demo1.ShortCut := ShortCut(vk_F9, []);
```

يعرض Delphi بصورة تلقائية الـ shortcut keys فى pop-up menu إلى اليمين من أمر الـ Demo . لا يجب عليك تعديل الـ caption الخاص ببند القائمة.

تعرف الاقواس الفارغة الـ null set لـ shift keys لتعيين Alt+F9، حدد قيمة TShiftState فى الاقواس:

```
Demo1.ShortCut := ShortCut(vk_F9, [ssAlt]);
```

لان shift state تعتبر Pascal set، يمكنك تحديد قيم متعددة منفصلة عن طريق الفصله. على سبيل المثال، هذه العبارة تعيين الـ Ctrl+Shift+A كاختصار (لايهم ترتيب القيم داخل المجموعة):

```
Demo1.ShortCut := ShortCut(Ord('A'), [ssCtrl, ssShift]);
```

لا يمكنك تحديد مفتاح الـ A باستخدم الـ virtual key code vk_A، بالرغم من ان هذه وغيرها من الـ codes الرقمية الابجدية الواردة فى online الخاص بـ Delphi وكذلك، ملف Windows.pas source code فى دليل Source\RTL الخاص بـ Delphi. استخدم Ord function الخاصة بالـ Pascal ورمز حرفى بين علامتى تنصيص فردية لتخصيص مفاتيح رموز الـ ASCII.

لتقسيم shortcut key إلى قيمة منفصله، استدع الـ ShortCutToKey، الذى يتم تعريفه بالآتى:

```
procedure ShortCutToKey(ShortCut: TShortCut;
  var Key: Word; var Shift: TShiftState);
```

قم بتمرير خاصية الـ ShortCut الخاصة بند القائمة إلى الـ parameter الأول. يعين الـ procedure، ShortCut's virtual key الخاصة بالـ hortCut's للـ Key والمجموعة التابعة لها من قيم للـ shift-state.

وتقوم function أخرى، وهى TextToShortCut ، بتحـويل قيـمـة الـ string إلى shortcut key . على سبيل المثال، خصص Alt+X بهذه العبارة :

```
Demo1.ShortCut := TextToShortCut('Alt+X');
```

يمكنك استخدام هذا الاسلوب لحث المستخدمين على إضافة shortcut key، ربما فى configuration utility تسمح المستخدمين ببرمجة اوامر القوائم الخاصة بهم :

```
var
  S: string;
begin
  S := InputBox('Input Dialog', 'Enter shortcut key', '');
    if Length(S) > 0 then
      Demo1.ShortCut := TextToShortCut(S);
end;
```

تعتبر وظيفة الـ InputBox فى متناول اليد لحث المستخدمين على عمل قيم الـ string والـ string الأول هو dialog's caption، والثانى هو استفسار معروض فى نافذة الـdialog والثالث (هنا بـ null) هو قيمة الـ string الأفتراضية وتعود القيمة بذلك اذا لم يدخل المستخدم قيمة الـ string .

قم باداء العملية العكسية مع الـ ShortCutToText ، التى تعرف بالآتى :

```
function ShortCutToText(ShortCut: TShortCut): string;
```

على سبيل المثال، يمكنك عرض shortcut key لعنصر قائمة كـ string فى dialog box :

```
ShowMessage('Demo1 shortcut = ' +
  ShortCutToText(Demo1.ShortCut));
```

تشغيل وابطال الأوامر:

لقد شرحت بالفعل فى هذا الباب كيفية تشغيل وابطال عناصر القائمة بتحديد خاصية الـ Enabled بـ True أو False . عندما تكون الـ Enabled محددة بـ False، يكون الـ caption الخاص ببند القائمة معتم فى نافذة الـ pop-up menu،

ولا يمكن للمستخدمين إختياره من الناحية العملية ، ان إعتام الاوامر الصحيحة فى الأوقات السليمة يتطلب تخطيطاً دقيقاً .

واسهل الطرق هو كتابة procedure بشكل قائمة بأكملها طبقاً للـ global flags والقيم الأخرى . على سبيل المثال ، قم بتعريف متغير Boolean FileIsOpen ، واكتب الـ procedure لتشغيل أو ابطال أوامر الـ Save والـ Open والأوامر الأخرى التابعة لقائمة File طبقاً لقيمة المتغير .

On the CD-ROM **على القرص المدمج:** ان تطبيق الـ ToDo على القرص المرفق بهذا الكتاب فى دليل الـSource\ToDo يوضح كيفية عمل هذا لقائمةFile عامة . والـ form class الخاصة بالبرنامج ، وهى TMainForm، تعرف procedure يسمىEnableMenu :.

```
procedure EnableMenu;
```

وينفذ البرنامج الـ procedure كما هو موضح فى شكل (٥-٣) . وتقوم العبارات بتشغيل وإبطال الاوامر فى قائمة File الخاصة بالبرنامج طبقاً لحالات حفظ الملف وفتح الملف التابعة للبرنامج ان نص هذه القائمة موجود فى ملف Main.pas الخاص بمشروع ToDo على القرص المدمج .

القائمة (٥-٣): الـ EnableMenu procedure من عينة الـ ToDo

```
applicationprocedure TMainForm.EnableMenu;
var
 I: Integer;
begin
 with FileMenu do
   begin
    for I := 0 to Count - 1 do       { Enable File commands }
      Items[I].Enabled := True;
      if not FileDirty then
    begin  { No edits }
     FileSave.Enabled := False;      { Must use Save as }
     if Length(Filename) = 0 then    { i.e. file not named }
```

```
      begin  { No edits; no name }
        FileSaveAs.Enabled := False;  { Nothing to save }
        FilePrint.Enabled := False;   { Nothing to print }
      end;
    end;
  end;
end;
```

برنامج الـ ToDo يستخدم flag وهو الـ FileDirty للإشارة اذا ما كانت هناك تغييرات قد حدثت للملف مفتوح. ويستخدم الـ EnableMenu procedure هذا الـ flag لتشغيل امر الـ Save الخاص بالقائمة، والذى يظل معتماً (وبذلك يبقى دون اختيار) اذا كانت بيانات الملف لم تتغير. ويقوم الـ procedure ايضاً بتشغيل وإبطال اوامر اخرى مثل Save As و Print .

لاستخدام الـ EnableMenu، قم بإنشاء event handler للـ OnClick event، الخاص بالـFileMenu، استدع الـ procedure. كنتيجة لذلك، يتم تشكيل اوامر القائمة قبل ان تصبح نافذة pop-up menu مرئية. على سبيل المثال، ها هو event handler للـ OnClick من الـ ToDo :

```
procedure TMainForm.FileMenuClick(Sender: TObject);
begin
  EnableMenu;  { Enable/disable commands before menu opens }
end;
```

فى تطبيق أكثر تعقيداً، قد يكون من الافضل ان تكتب procedure لتشغيل قائمة لكل من قوائم البرنامج، File، Edit ، و Window وغيرها. استدع الـ procedures من الـ OnClick لكل بند MainMenu . رغم ذلك، هذا لا يمنع المستخدمين من اختيار اوامر بضغط shortcut key . والسبيل الأكثر امناً هو ان يؤدى كل امر الصلاحيات الخاصة به. على سبيل المثال، امر الـ File|Save قد يخرج ببساطة دون اتخاذ أى عمل اذا لم يكن هناك ملفاً مفتوحاً، وبذلك، لا يوجد شئ يراد حفظه.

دمج وفصل للـ MainMenu objects:

يمكن لأى form ان يكون لها MainMenu objects متعددة والتى يمكنك دمجها فى بعضها لإنشاء قوائم ديناميكية . وتعتبر هذه التقنية مفيدة بالأخص فى البرامج ذات two forms أو أكثر . عندما يعرض البرنامج form ثانوية ، تنتج أوامر الـ MainMenu الخاصة بها تلقائياً مع النافذة الرئيسية للبرنامج . على سبيل المثال ، يمكن للـ configuration dialog أن يستخدم هذه التقنية لدمج مجموعة جديدة من الأوامر فى menu bar . عندما يغلق المستخدم الـ dialog تختفى الأوامر المدرجة بصورة تلقائية .

دمج MainMenu objects:

إن العنصر الرئيسى فى القوائم المدرجة هو قيمة الـ GroupIndex Byte فى الـ TMenuItem class . وهذه القيمة ، والتى قد تتراوح من صفر إلى ٢٥٥ ، تحدد تأثير الدمج . استخدم قيم GroupIndex المساوية لإحلال إحدى القوائم مكان أخرى . استخدم قيم أقل لإدخال قوائم إلى اليسار . استخدم قيم أعلى لإدخال قوائم إلى اليمين .

قم بتعيين قيم GroupIndex فى مضاعفات الـ ١٠ لتيسير عملية إدخال قوائم جديدة بين القوائم الموجودة . ولكن ، فى تطبيقات الـ OLE (أنظر الباب السادس عشر) ، يتطلب الـ Windows منك أن تستخدم قيم GroupIndex محددة لتشغيل دمج القوائم in-place editing .

اتبع هذه الخطوات لدمج MainMenu objects فى نفس الـ form ولتجربة هذه التقنية :

١ - أضف الـ two MainMenu objects على الـ form إننى استخدم اسماء الـ object الإفتراضية وهى MainMenu1 و MainMenu2 ، ولكن يمكنك تعيين هذه الاسماء اذا أردت .

٢- اضغط مرتين الـ MainMenu1 واستخدم الـ Menu Designer لإنشاء قائمة Demo ذات أمر الـ Advanced . يقوم Delphi بتسمية عنصر القائمة بـ Demo1 طبقاً للنظام الإفتراضى . اختر Demo1 فى نافذة الـ Object

Inspector ، وحدد قيمة الـ GroupIndex الخاصة بـ Demo1 بصفر (القيمة الإفتراضية).

٣- اضغط مرتين MainMenu2، وقم بإنشاء قائمة Extra ذات أمر Exit. يقوم Delphi بتسمية بند القائمة بـ Extra1 طبقاً للنظام الإفتراضى. اختر Extra1 فى نافذة الـ Object Inspector ، وحدد قيمة الـ GroupIndex الخاصة بـ Extra1 بواحد.

٤- قم بإنشاء الـ event handler للـ Advanced command بإختيار فى الـ form أو فى الـ Menu Designer، وأدخل العبارة التالية بين البداية والنهاية :

```
MainMenu1.Merge(MainMenu2);
```

٥- قم بإنشاء الـ event handler لأمر الـ Exit. (لأن الـ MainMenu2 غير ظاهرة فى الـ form، يجب أن تستخدم الـ Menu Designer لإختيار الأمر). أدخل عبارة Close للـ procedure الخاص بهذا الأمر.

٦- اضغط F9 لتشغيل البرنامج. اختر أمر الـ Demo|Advanced لدمج القائمة الثانوية، وجعل أمر الـ Exit متاحاً.

قم بتجربة قيم GroupIndex مختلفة. على سبيل المثال، حدد الـ GroupIndex بواحد لكل من Demo1 و Exit1. إن إختيار أمر الـ Advanced عندئذ يجعل قائمة الـ Extra تحل محل الـ Demo. أو، حدد الـ GroupIndex بقيمة أعلى فى الـ Demo1 منها فى الـ Exit1. يقوم أمر الـ Advanced الآن بإدخال قائمة الـ Extra إلى اليسار من Demo.

فصل الـ MainMenu objects:

قم بإبطال تأثير الدمج بإستدعاء UnMerge method لأى MainMenu object. على سبيل المثال، أضف زراً على نافذة الـ form واستخدم العبارة التالية فى الـ OnClick الخاص بالزر. إن ضغط الزر يحذف الـ MainMenu2 من الـ MainMenu1:

```
MainMenu1.UnMerge(MainMenu2);
```

دمج MainMenu objects في forms متعددة:

عند دمج MainMenu objects في two forms أو أكثر، استخدم خاصية الـ AutoMerge لتحديد ما اذا كان الدمج يتم بصورة تلقائية أو يحدث بموجب تحكم البرنامج. قم بتجربة هذه الخطوات لتجربة دمج قوائم forms متعددة:

١- أضف MainMenu object في الـ form، اضغطه مرتين، استخدم الـ Menu Designer لإنشاء قائمة Demo ذات أمر واحد، وهو الـ Show form. مازالنا في الـ Menu Designer، اضغط مرة واحدة قائمة Demo، ثم استخدم الـ Object Inspector لتحديد الـ GroupIndex بصفر (القيمة الإفتراضية). لاحظ أن Menu object يسمى Demo1 وهو object من TMenuItem class. تأكد من إختيار الـ menu objects باستخدام الـ Menu Designer، وليست الـ form التي تظهر بها القائمة.

٢- أضف form خالية ثانية إلى المشروع باستخدام أمر الـ File|New Form. قم بإعادة تحديد حجم نافذة هذه الـ form، والتي تسمى Form2، وحركها جانباً.

٣- في الـ Form2، أضف الـ MainMenu object منفصل، وحدد خاصية الـ AutoMerge التابعة له بـ True. لاحظ أن الـ object يسمى MainMenu1، مثل MainMenu object الآخر. ولأن الـ objects تقع في forms منفصلة، لا يوجد تداخل بين الاسماء. اضغط مرتين الـ MainMenu1 object التابع لـ Form2 واستخدم الـ Menu Designer لإنشاء قائمة Advanced ذات أمر Close. اختر هذا الأمر وأدخل عبارة ;Close في الـ event handler.

٤- وبينما مازالنا في الـ Form2 ونستخدم الـ Menu Designer، اضغط مرة واحدة بند قائمة الـ Advanced وحدد خاصية الـ GroupIndex الخاصة بها بواحد. (اذا لزم الأمر، اضغط مرتين الـ MainMenu1 object في Form2 للرجوع للـ Menu Designer). إنتقل إلى Form1، اختر أمر الـ Demo|Show form، وأدخل هذه العبارة في الـ event handler الناتج.

```
Form2.Show;
```

٥- أضف تعريف الـ uses تحت الكلمة الرئيسية Unit1 implementation مباشرة:

```
uses Unit2;
```

٦- افعل نفس الشئ لـ Unit2، ولكن قم بالإشارة إلى Unit1 بالآتى (كل unit تستخدم الأخرى حالياً):

```
uses Unit1;
```

٧- أخيراً، قم بإنشاء الـ event handler للـ OnClose event الخاص بـ Form2. أدخل هذه العبارة فى الـ procedure.

```
Form1.MainMenu1.UnMerge(MainMenu1);
```

٨- اضغط F9 لتشغيل البرنامج. اختر أمر الـ Demo|Show form، ولاحظ ان النافذة الرئيسية لها الآن عنصر قائمة Advanced. اختر أمر الـ Close الخاص بهذه القائمة لإغلاق النافذة الثانوية واستعادة القائمة المدمجة فى تنسيقها الأصلية.

قد تبدو الخطوات السابقة معقدة بعض الشئ فى المرة الأولى التى تقوم بها. ولكن هذه التقنية مفيدة عندما يملى عليك مظهر نافذة مستوى فرعى تغييراً لمجموعة القوائم الخاصة بالبرنامج- على سبيل المثال، عند برمجة نافذة Find dialog. عند دمج MainMenu objects فى forms متعددة، تذكر هذه الأفكار:

- أضف MainMenu object واحداً على كل form. حدد الـ AutoMerge بـ False للـ main window's object. حدد الـ AutoMerge بـ True لكل MainMenu object الأخرى. اذا كنت لا تريد دمجاً بصورة بالقائمة، حدد الـ AutoMerge بـ False لكل القوائم، واستدع Merge لأداء دمج القوائم كما فعلت فى حالة الدمج فى form واحدة.

- أضف اسماء unit module ثانوية (Unit2، Unit3، وما إلى ذلك) إلى القطاع uses فى Unit1. وهذا يخبر الـ module الرئيسية ان تستخدم الـ modules الثانوية.

- أضف اسم الـ main module الرئيسية (Unit1، مثلاً) لكل قطاع uses جديد فى كل modul ثانوية.

• استدع UnMerge للـ MainMenu object الخاص بالـ form الرئيسية لإزالة قائمة بصورة تلقائية عند إغلاق الـ form الثانوية . انظر الخطوة رقم ٧ فى البرنامج التعليمى السابق. بالرغم من تحديد الـ AutoMerge ، لا يمكن ان يكون الفصل بصورة تلقائية عند إغلاقة الـ form (رغم انه ربما كان يجب ان يكون كذلك) .

• استدع الـ Show لعرض نافذة الـ form الثانوية الـ child أو الـ modeless dialog للنافذة الرئيسية الأم .

• و Delphi يجعل اسم كل MainMenu object فى كل form ، MainMenu1 . للحصول على برنامج اوضح ، اختر اسماء افضل لهذه الـ objects مثل MainFormMainMenu و FindFormMainMenu .

تتطلب تطبيقات الـ OLE والـ MDI تقنيات مختلفة لدمج القوائم . انظر الباب الخامس عشر والسادس عشر لمزيد من المعلومات حول برمجة الـ OLE والـ MDI .

تعديل قائمة النظام:

تظهر قائمة النظام عندما تضغط ايقونة البرنامج فى الركن الايسر العلوى بالنافذة . لا يجب ان تحتاج معظم التطبيقات إلى إضافة اوامر لهذه القائمة- وكذلك ، يتعامل Delphi بصورة تلقائية مع الاوامر المعيارية للقائمة . ولكن ، اذا اردت تعديل قائمة النظام لتطبيق معين ، فنحن نشرح لك كيفية ذلك .

تعتبر قائمة النظام ملكاً للـ Windows ، وليس للتطبيق ولذلك ، لا يمكنك استخدام MainMenu object للتوصل إلى قائمة النظام . لإضافة أمراً ، استدع الـ Windows AppendMenu . للإستجابة إلى اختيار الأمر ، اضف message-handler procedure للـ form's class . اتبع هذه الخطوات لتجربة التقنية واضف أمر الـ ...About لقائمة النظام :

١- قم بإنشاء الـ handler للـ OnCreate event الخاص بالـ from الرئيسية . ادخل البرمجة للـ FormCreate من القائمة (٥-٤) "هذا النص موجود على القرص المدمج فى ملف Main.pas لمشروع الـ SysMenu) .

٢- قم بتعريف رقم ثابت ليمثل الأمر الجديد . استخدم أى قيمة صغيرة مثل الكسر السداس $00A0 (١٠٠ عشر) . يستخدم الـ Windows البت الاربع القلائل لهذه القيمة للـ command flag والتى يجب ان تكون صفراً . واسهل طريقة لتلبية هذا المتطلب هى تعيين قيم . فى الكسر السداسى مع آخر رقم مساوياً لصفر . على سبيل المثال ، اضف هذا التعريف للـ unit's implementation :

```
const
    cm_About = $00A0;
```

٣- فى الـ form's class (وهى TForm1 طبقاً لنظام الافتراضى) ، ادخل تعريف الـ message-handler التالى فى الـ class's private . والسطر الأول يعرف الـ procedure ، والذى يجب ان يكون به الـ procedure المتغير الفردى TWMSysCommand كما هو موضح . والسطر الثانى يخبر Delphi ان تستدعى هذا Delphi عندما تتلقى الـ form رسالة wm_Syscommand من الـ Windows . تشير هذه الرسالة إلى ان المستخدم قد إختار أمر قائمة نظام :

```
procedure WMSysCommand(var Message: TWMSysCommand);
  message wm_SysCommand;
```

٤- قم بتنفيذ الـ WMSysCommand كما هو موضح فى القائمة (٥-٤) . قم بتشغيل البرنامج واختر أمر الـ ...About الخاص بقائمة النظام لعرض display Box .

فى القائمة (٥-٤) ، تظهر الـ WMSysCommand كيفية التعامل مع رسالة فى form class . يتلقى الـ procedure متغير TWMSysCommand الذى يصف محتويات الرسالة . فى هذا المتغير ، ليمثل مجال الـ CmdType قيمة الامر الذى تم اختياره . اذا كانت هذه القيمة تساوى ثابت الـ cm_About الخاص بنا ، تستدعى عبارة case الـ ShowMessage لعرض .dialog box . لأداء الانتاج الافتراضى للأوامر الاخرى لقائمة النظام ، يستدعى الـ procedure inherited لقيم الـ unhandled CmdType .

ويعرض الـ OnCreate الخاص بالـ form كيفية إيجاد للـ window handle لقائمة النظام . يخزن الـ procedure هذا الـ handle فى متغير MenuH ، من نوع الـ HMenu (menu handle) . أولاً الـ GetSystemMenu يعيد الـ menu

handle لنافذة الـ form- وقيمة الـ Handle المتغيره في هذه العبارة تنتمي لل form. وتضيف Windows AppendMenu function أمر أمر الـ About... لهذه القائمة.

القائمة (٥-٤): إضافة اوامر لقائمة نظام التطبيق

```
procedure TForm1.WMSysCommand(var Message: TWMSysCommand);
begin
  case Message.CmdType of
      cm_About: ShowMessage('About command selected!');
      else
      inherited;  { Default processing }
      end;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  MenuH: HMenu;
begin
  MenuH := GetSystemMenu(Handle, False);
    AppendMenu(MenuH, mf_String, cm_About, 'About...');
end;
```

بعد ذلك، يكرر OnCreate الخطوتين السابقتين ولكن، هذه المرة، يقوم البرنامج بتمرير الـHandle من الـ application object إلى الـGetSystemMenu. وهذا يعتبر ضرورياً لتعديل قائمة النظام لتطبيق بعيداً عن الرؤية.

:Portable Menus

كثيراً من قوائم التطبيقات لها اوامر متشابهة، ويمكنك توفير الوقت بتطوير portable menus للتشارك فيما بين برامج متعددة. يتم استخدام أداتان أساسيتان لإنشاء الـ portable menus: الـ resources والـtemplates .

:Menu templates

ان Delphi لديه menu template منشأه متعددة يمكنك إدخالها في الـ PopupMenu objects في الـ MainMenu. لإدخال template:

١- أضف واحداً من الـ two objects على الـ form .

٢- اضغطه مرتين لفتح الـ Menu Designer .

٣- اضغط زر الفأرة الأيمن أو اضغط Alt+F10 لفتح floating pop-up menu .

٤- اختر Insert From Template .

٥- اختر موضع قائمتك (قائمة File للتخزين على سبيل المثال) .

بعد إضافة الـ menu template من الصواب ان تعيد تسمية كل الـ item objects للقائمة طبقاً للإرشادات التي ذكرتها . هذا يستغرق لحظات معدودة ، ولكن يساهم بشكل كبير فى الحصول على code يمكن قراءتها . افعل هذا دائماً قبل كتابة عبارات تشير إلى objects باسماءها . ان الاسماء الأفتراضية الخاصة بالـ Menu Designer ، مثل Edit1 و Save1 ، لا تعتبر واصفة بالقدر الكافى . ومن الأفضل التسمية بـ FileEdit و FileSave للإشارة إلى أى قائمة ينتمون . ويعتبر FileEditItem ، و FileSaveItem اسماء افضل من السابقة لأنها تذكرك بأنها objects من TMenuItem .

فكرة: اذا قم باضافة menu template إلى PopupMenu object ، يقوم Delphi بإنشاء قائمة متداخلة وكأن الـ floating pop-up menu عبارة عن menu bar إلتف على جانبه . جرب هذا ! هذه التقنية الجيدة لواجهة التطبيق لا تشغيل فى اغلب الاحيان .

يمكنك ايضاً حفظ أى قائمة على انها template استخدام امر الـ Save as Template الخاص بالـ Menu Designer لتخزين القوائم التى تفضلها مع مخزون الـ template . يخزن الـ Delphi الـ menu templates فى ملف Delphi32.dmt ، الموجود فى دليل bin\ الخاص بـ Delphi .

ولا يتم تخزين الـ event handlers وخصائص الاسم مع menu templates لان هذه فى الغالب ستكون مختلفة فى كل تطبيق . و menu templates تعتبر الـ MenuItem objects خالصة . فهى لا تحتوى على ايه code . اذا اردات ان تحفظ الـ code مع menu template قم بإنشاء برنامج

اختياري لكل قائمة، ثم بعد ادخال الـ template في تطبيق جديد، قم بنسخ ولصق عبارات من الـ module. المختبره.

Menu resource scripts :

والطريقة الثانية لإنشاء portable menus هي تصميمها بالطريقة القديمة، باستخدام resource script command . يمكنك ايضاً استخدام هذه التقنية لإدخال قوائم من تطبيقات اخرى- كتلك التي تم تطويرها في الـ C أو الـ C++ ، مثلاً.

على القرص المدمج: يوضح القائمة (٥-٥)، والقائمة (٥-٦) menu script . والـ Multimen.inc يعتبر include File يعرف الثوابت لتمثيل بنود القائمة. الـ Multimen.mnu يعتبر resource script يعرف تخطيط في نص القائمة وLayout. يمكنك العثور عليهما في الدليل الفرعي Multimen على القرص المدمج المرفق.

القائمة(٥-٥):Multimen.inc

```
const
 id_Menu       = 100;
 cm_FileMenu  = 101;
 cm_FileExit    = 102;
```

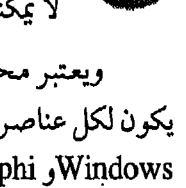
القائمة (٥-٦): Multimen.mnu

```
#include multimen.inc

id_Menu MENU
BEGIN
 POPUP "&File"
   BEGIN
    MENUITEM "E&xit", cm_FileExit
    END
   POPUP "&Edit"
   BEGIN
    MENUITEM "&Undo", 201
```

```
    MENUITEM "&Redo", 202
      POPUP "&Search"
      BEGIN
        MENUITEM "&Ignore case", 301, CHECKED
        POPUP "&Find"
        BEGIN
          MENUITEM "&Previous", 401
          MENUITEM "&Whole words", 402
          MENUITEM "&Partial words", 403
        END
        MENUITEM "&Replace", 303
      END
    END
END
```

لاســــتــــخـــــدام نص الـ Multimen.mnu، أضـف MainMenu أو PopupMenu على الـ form. اضــغط الـ object مــرتين لفــتح الـ Menu Designer، واخــتـر امـر الـ Menu Designer، واخـتـر امـر الـ Insert From Resource. ادخل ملف نص الـ mnu، (بالمنسبة، إنها الطريقة التى أنشأت بها القائمة المتداخلة لتطبيق الـ MultiMen فى هذا الباب).



على القرص المدمج: بعـد إدخـال نص قـائمـة فى تطبـيق Delphi، يمكنك استخدام الـ Menu Designer لتعديل اوامر قائمتك. ولكن، لا يمكنك بسهولة تعديل النص الأصلى ودمجه مرة اخرى فى التطبيق.

ويعتبر محمل نص القائمة الخاص بـ Delphi غير معيارى نوعاً ما. يجب ان يكون لكل عناصر القائمة على menu bar بند امر واحد على الأقل. ولكن تطبيقات Windows و Delphi تسمح بالأ يكون لعناصر القائمة أوامراً على الـ menu bar. ان اختيار بند بلا أمر لا يؤدى إلى فتح نافذة قائمة ولكن ينفذ فوراً العنصر على أنه أمر.

لايســمح الـ Menu Designer بـ command-less menu resources ولذلك، لا يمكن ان يكون للـ resource script اثنين من اوامــر الـ POPUP متتابعين، كما فى هذا المثال:

```
POPUP "&Demo"
POPUP "&Help"
```

وتعتبر القوائم التى ليست لها أوامر قوائم غير معتادة، ولذلك تعتبر هذه المشكلة صغيرة. ولكن، يقدم Borland Pascal 7.0 واغلب نظم تطوير الـ C والـ C++ قوائم بلا أمر، وقد تجد resource script من حين لآخر لا يستطيع Delphi قراءته. يمكنك ابقاء المشكلة بإضافة كلمات BEGIN و END رئيسية لقوائم الـ POPUP. استخدم code مثل:

```
POPUP "&Demo"
BEGIN
END
POPUP "&Help"
BEGIN
END
```

افكار عن الـ Menu Designer:

ان الـ Menu Designer الخاص بـ Delphi له واجهة تطبيق فطرية يسهل التحكم فيها. ولكن، فيما يلى بعض الأفكار التى قد لا تكون واضحة:

- اضغط Ins لإدخال عنصر قائمة خالية أعلى العنصر الذى يتم ابرازه حالياً. قم بابراز عنصر الـmenu واضغط Ins لإضافة pop-up menu فى الـ menu bar.

- اضغط Del لحذف عنصر قائمة. كن على حذر عند ضغط Del- لا يمكنك ابطال حذف قائمة.

- أضف هذه العلامة (-) فى الـ Caption الخاص بالقائمة لتكوين خط فاصل. يظهر هذا الخط بالحجم المناسب للـ form وفى وقت التشغيل، ولكن فى الـ Menu Designer، يكون الفاصل بطول أمر قائمة طبيعى بحيث يمكنك اختياره وتحريره كأى بند قائمة آخر.

- لإنشاء قائمة متداخلة، قم بابراز أى عنصر قائمة واضغط Ctrl + السهم المشير لليمين اضغط Esc للعودة إلى المستوى السابق، أو استخدم الفأرة ومفاتيح الاسهم للملاحة عبر القائمة.



● اسحب واسقط عناصر القائمة لإعادة ترتيبها . جرب هذا مع القائمة إنها اسهل طريقة لأداء تغيير كبير على تخطيط قائمة .

ActionLists:

وواجهة التطبيق الخاصة بالمستخدم توفر طرق عديدة لأداء نفس الوظائف . إنها تزود المستخدمين بمستويات مهارة مختلفة ، ويمكن ان يقلل من عبء تعلم كيفية استخدام تطبيق جديد على سبيل المثال ، بدلاً من استخدام اوامر القائمة فقط لتحريك برنامج ، يمكن لواجهة التطبيق الجيدة ان توفر ازرار وtoolbars لتنفيذ نفس الأوامر .

ولكن بالنسبة لواضعى البرامج ، كلما زادت المسارات إلى اهداف واجهة التطبيق ، زاد العمل الذى يجب القيام به لإنشاء code يمكن الاعتماد عليها . فى برنامج ذى عدة عشرات من الأوامر ، فإن إضافة ازرار toolbar تتشارك فى نفس الـ code يعود إلى حد كبير للتفاعل بين هذه الأوامر وحالة البرنامج . واحد الامثلة على ذلك هو أنه من الصعب ابطال الأوامر الغير مناسبة بشكل سليم فى الاوقات الصحيحة . وكذلك يصعب تقليل الازدواج فى الـ source code- على سبيل المثال ، ان كتابة نفس العمليات لـ two command objects مختلفان بطريقتين مختلفتين ، والتى قد تؤدى إلى اعطاب فى المستقبل .

يقدم Delphi مفهوم الـ action list للمساعدة على حل هذه المشكلات . ان الـ ActionLIst الموجودة فى Standard palette يمكن ان يخزن Action object واحد أو أكثر . ويعد كلobject من Action objects نوعاً من المرساة التى تتعلق بها الـobjects الأخرى . لتوفير events وخصائص عامة لإثنين أو أكثر من الـ objects الأخرى مثل بنود القائمة والازرار ، فعليك ان تبرمج الـ Action ليفعل ما تريد به وتلحقه لـ objects الأخرى . وكلاً من هذه الـobjects يتخذ سمات Action component وكأنك برمجت كلاً على حدى .

Note

ملاحظة: ان الـ ActionList قد تم برمجته فى TActionList class . يعتبر الـ ActionList من TActionList class والـ ActionList فقط هو الذى يظهر على palette- يتم إنشاء Command objects بواسطة action list editor ، والتى يوضح هذا الفصل كيفية استخدامها وهذا الترتيب

يشـبـه الـطريقـة التى يخـزن بهـا الـ MainMenu component object الـ MenuItems الفردية .

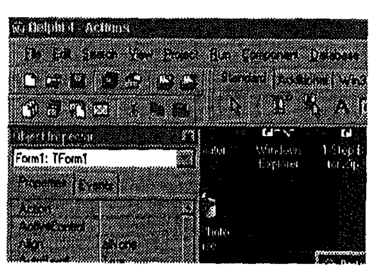
ان مـشـروع الـ Action فى القـائمـة (٥-٧) يوضـح كـيـفـيـة برمـجـة CommandList . يوضح شكل (٥-٥) مظهر وقت التصحيح الخاص بالبرنامج داخل Delphi . ويظهـر هذا البـرنامج كـيف يخـزن الـ ActionList الـ Action object والتى تقدم صفات عامة وبرمجة لإثنين أو أكثر من الـ objects الأخرى . لتشغيل البرنامج ابتع هذه الخطوات :

١ - افـتـح مـشـروع الـ Action باسـتـخـدام Delphi (يجب ان يكون لديك Version 4 لهذا العرض) .

٢ - قم بتشغيل البرنامج بضغط F9 .

٣ - أدخل Quit فى edit box لتشغيل زر Exit . والأمر الذى يحمل نفس الأسم .

٤ - يمكنك اختيار إما بند القائمة أو ضغط الزر لإنهاء البرنامج والعودة إلى Delphi .



شكل (٥-٥) : أدخل Quit فى edit box لتشغيل زر Exit لخاص بالبرنامج. هذا يظهر كيفية استخدام الـ ActionList

القائمة (٥-٧) : Actions\Main.pas

```
unit Main;

interface

uses
```

```
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
   Forms, Dialogs, Menus, ActnList, StdCtrls;

type
 TForm1 = class(TForm)
    MainMenu1: TMainMenu;
     Edit1: TEdit;
     Button1: TButton;
     Label1: TLabel;
     ActionList1: TActionList;
     Demo1: TMenuItem;
     Exit1: TMenuItem;
     ExitAction: TAction;
     procedure ExitActionExecute(Sender: TObject);
     procedure ExitActionUpdate(Sender: TObject);
   private
    { Private declarations }
    public
  { Public declarations }
    end;

var
 Form1: TForm1;

  implementation

  {$R *.DFM}

  procedure TForm1.ExitActionExecute(Sender: TObject);
  begin
   Close;  // Exit program
  end;

  procedure TForm1.ExitActionUpdate(Sender: TObject);
  var
   Flag: Boolean;
```

```
    begin
    // Set Flag True if user types Quit into Edit1
      if Lowercase(Edit1.Text) = 'quit'
         then Flag := True
           else Flag := False;
         Button1.Enabled := Flag;  // Enable or disable Button1
      Exit1.Enabled := Flag;    // Enable or disable Menu item
  end;
```



على القرص المدمج: لتعليم كيفية استخدام الـ ActionList اتبع هذه الخطوات لإعادة إنشاء تطبيق الـ Action على القرص المدمج فى دليل الـ Source\Actions :

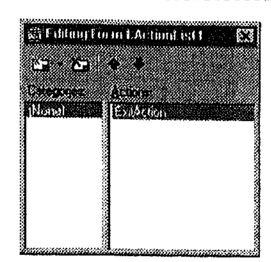
١- ابدأ تطبيقاً جديداً. أضف MainMenu، Edit، Button، Label، و ActionList واحداً فى كل مرة.

٢- اضغط مرتين الـ MainMenu1 وقم بإنشاء قائمة Demo مع أمر الـ Exit. وهذا ينشئ الـ Exit1، والذى لا يظهر فى الـ form.

٣- حدد خصائص الـ Enabled بـ False للـ Exit1 وللـ Button1 (اختر الـ Exit1 باستخدام قائمة اللائحه الخاصة بالـ Object Inspector). هذا يحدد الخصائص الأولية بـ two command objects، عند تشغيلهما عبر shared ActionList، ينهى البرنامج.

٤- اضغط مرتين الـ ActionList. مثل الـ MainMenu والـ nonvisual components الأخرى، يظهر الـ ActionList كأيقونة على الـ form، ولكنه غير مرئى فى وقت التشغيل. إنك ترى الآن action list editor التى يمكنك ان تنشئ فيها shared Action [انظر شكل (٥-٦)].

٥- اضغط زر الـ New Action (الأول على اليسار) لإنشاء الـ Action. وهذا يسمى Action1 حسب النظام الافتراضى. فى هذا العرض، نستخدم Action واحد فقط، ولكن يمكنك إضافة أى عدد من الـ Action تحتاجه الـ ActionList الواحد.



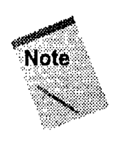
شكل (٥-٦): action list editor الخاص بـ Delphi 4
لموضح هنا، اضغط مرتين لـ ActionList

٦- ان استـخـدام قائمـة لائحـة الـObject Inspector يخـتـار الـ Action1 الذى تم إنشاءه فى الخطوة السابقة. (وهذا قد تم اختياره فى الغالب) سوف تحتاج بطبيعية الحال إلى تغيير اسم هذا الـ object تعكس هدفه. فى هذه الحالة، قم بتغيير خاصية الـ Name إلى الـ .ExitAction وحدد خاصية الـ Caption بـ E&xit .

٧- اضـــغـط Events tab الخــاص الـ ExitAction فى الـ Object Inspector. اضـغط مـرتين كلاً من OnExecute و OnUpdate، لإنشـاء event handler. إملا هذه الـ procedure باستـخدام القـائمـة كمرشـد لك- إننى سـوف اشرح البرمجة بعد هذه الخطوات.

٨- والخطوة الأخـيـرة فى عـمليـة إنشـاء ActionList هى ربط الـ controls التى تتشـارك فىevents وخصـائص الـ Action. لعمل هذا، اختـر الـ Exit1، باسـتـخدام الـ Object Inspector drop-dow List، وحـدد خـاصـيـة الـ Action له بـ ExitAction. بنفس الطريقة، اختر الـ Button1. حدد خاصية الـAction له ExitAction ايضاً يتشارك الآن الزر وبند القـائمـة فى الـevents وخصـائص الـ ExitAction. لاحظ ان الـ Captions الخاصة بالـobject المشترك تتغير إلى خاصية الـ Captions التابعة للـ ExitAction.

٩- قم بتشغيل البرنامج بضغط F9. ادخل Quit فى edit box لتشغيل الـ command objects، والتى عند اختـيـارها، تنفـذ الـ OnExecute event للـ ExitAction المشترك، وتنهى البرنامج.





ملحـوظة: تعـتـبر خـاصـيـة الـ Action، من TAction class، جـديدة فى Delphi، إنهـا مـقـررة كـعـضـو خـاص فى classes أخـرى مـتنوعـة مـثل MenuItems، Button، BitBtn، SpeedButton، components أخرى.

لكل Action object مثل الـ ExitAction في برنامج العرض two events: OnExecute و OnUpdate . ويقدم OnExecute برمجة تنفذ عندما يتم اختيار أي object آخر مثل الـ Button أي الـ MenuItem . وتلك الـ objects الأخرى تحدد خاصية الـ Action الخاصة بها بـ ExitAction المرتبط بالـcode المشتركة، والتي، في هذه الحالة تنهي البرنامج باستدعاء Close .

بطريقة مشابهة، يقدم الـ Actions OnUpdate الفرصة لتغيير أكثر من الـobjects مشتركة . في برنامج العرض، وقمت ببرمجة OnUpdate الخاص بالـ ExitAction لتحديد خصائص الـ Enabled للـ Exit1 والـ Button1 بـ True أو False ، إعتماداً على اذا ما كانت خاصية الـ Text للـ Edit1 تساوي "quit" . يتم استعاء الـ "quit" . يتم استدعاء الـ OnUpdate تكراراً في أثناء ما يكون (لبرنامج غير مشتغل، لذا لا تضيف برمجة مطولة هنا . إن تحديد خصائص الـ True أو False كما حدث في العرض يعتبر الاستخدام الأمثل لـ OnUpdate الخاص بالـ Action object .

Tip

فكرة: قم بإضافة ActionList object في الـ data module (أنظر الباب الثالث، معلومات حول الـ Form). هذا يقلل التزاحم في الـ forms المرئية، وكذلك يوفر توصلاً سهلاً للـ ActionList والـ Action objects الخاصة به إلى الـ modules الأخرى . ببساطة استخدم data module في أي form .

افكار للمستخدم الخبير

● بالرغم من أن الـ MenuItem تعد component ، فلا يمكنك إختيارها جميعاً في نافذة الـ form لبرمجة خصائص و events مشتركة . لإنشاء event مشترك بين MenuItem objects متعددة، اكتب procedure واحداً للبند الأول أو أي بند آخر، وعينة للـ objects الأخرى واحداً تلو الآخر في نافذة الـ Object Inspector . لا يبدو أن هناك طريقة أسهل لفعل هذا . وبالتبادل، يمكنك تعيين procedures للـ OnClick الخاص بالـ objects في وقت التشغيل- ربما في برنامج OnClick event handlers الخاص بالـ form .

● قم بزيادة مجالات الـ Tag فى MenuItem OnClick event handlers الخاصة بالـ object's اطبع هذه القيم قبل أن ينتهى البرنامج . استخدم هذه البيانات لترتيب أفضل لعناصر القائمة- وضع أوامر يتم استخدامها بصورة مكثفة على القمة، مثلاً .

● أدخل نسخة من أمر القائمة الذى تم إختياره على قمة قائمة . وعندما يختار المستخدمون الأوامر، فإنها تظهر أكثر قرباً من قمة نافذة القائمة . قد تستخدم هذه التقنية لعمل أوامر يتم إختيارها تكراراً ويمكن التوصل إليها- مثل اسماء للـ fonts .

● فى الـ Menu Designer، قم بتحديد خاصية الـ Break لـ MenuItem object بـ mbBarBreak أو بـ mbBreak لإنشاء horizontally segmented pop-up menus . أنظر القائمة الناتجة فى الـ form- إنك لا ترى تغييراً مرئياً فى الـ Menu Designer . وهذه السمات نادراً ما تستخدم فى برمجيات الـ Windows، ولكنها متوفرة لإقامة بعض الأنواع من القوائم المعقدة، وخاصة تلك التى بها أوامر عديدة والتى تجعل نافذة القائمة تمتد بعد الحد السفلى للنافذة .

● استخدام خاصية الـ Alignment فى الـ PopupMenu objects لتغيير مكان نافذة القائمة بالنسبة لمؤشر الفأرة . على سبيل المثال، حدد الـ Alignment بـ paCenter لوضع قمة النافذة فى مركز مؤشر الفأرة . يعمل الـ Alignment مع floating pop-up menus فقط المعينة لخاصية الـ PopupMenu التابعة للـ form .

● يوفر الـ ActionList خاصية الـ Images التى يمكنك استخدامها لتخزين BitBtns لكى تعرض بواسطة visual objects أخرى مثل الـ MenuItems والـ BitBtns . أضف الـ Images الخاصة بك إلى الـ ActionList . لربط كل صورة للـ Action components الفردية، استخدم خاصية الـ ImageIndex التابعة Action object . وهذه الخصائص تجعل من السهل ربط الـ bitmaps بـ components متعددة .

المشروعات التى يمكنك تجربتها



٥-١ : اكتب برنامجا للـ standard File menu template . أضف code من هذا الباب كتنفيذ أوامر الـ Open، الـ Save، الـ Save As،

والأوامر الأخرى. وكذلك أضف code من التطبيق FileMenu لإلحاق اسماء ملفات إلى أسفل القائمة. احفظ الناتج كإحتياطى لبدء مشروعات جديدة متصلة بالملف.

٥-٢ : قم بتعديل التطبيق FileMenu للسماح للمستخدمين بتعيين عدد الاسماء المعروضة فى قائمة الـ File.

٥-٣ : متقدم: اكتب برنامجا يسمح للمستخدمين ببناء قائمة جديدة فى وقت التشغيل. ضمنه أوامراً لإدخال accelerator shortcut keys.

٥-٤ : متقدم: باستخدام الـ code الخاص بك من مشروع (٥-٣)، اكتب قوائم بصورة resource script للإدخال فى تطبيقات Delphi.

٥-٥ : أعد برمجة Menu Designer's templates لاستخدام TMenuItem object أكثر وصفاً. لعمل ذلك، أضف MainMenu object على form ثم اضغطه مرتين، وقم بتحميل كل template فى Menu Designer. قم بتعديل اسماء الـ object. بإختيارها وتغيير خصائص الـ Name لها فى الـ Object Inspector. على سبيل المثال، يمكنك تغيير الـ Save1 التابع لأمر الـ FileSaveItem. وأخيراً، باستخدام الـ Menu Designer، احفظ الـ template تحت نفس الأسم أو إسماً مختلفاً.

٥-٦ : اعد برمجة مشروع الـ Actions. قم بتحويل الـ ActionList object إلى data module (انظر الباب الثالث). قم بتعديل الـ Button objects والـ MenuItem للوصول إلى الـ ExitAction الخاص بالقائمة، على سبيل المثال، على أنه DataModule1.ExitAction.

ملخص:

- استخدم الـ MainMenu object لإنشاء menu bar نافذة رئيسية.
- استخدم الـ PopupMenu لإنشاء floating pop-up menus تظهر عندما يضغط المستخدمون زر الفأرة الأيمن. يمكنك أيضاً استدعاء الـ Popup method لعرض floating pop-up menus فى أى موضع على الشاشة.

• تعتبر جميع العناصر في الـ MainMenu objects والـ PopupMenu ، objects في TMenuItem class . قم بتعيين قيم لخصائص الـ CheckMark ، الـ Visible ، والـ Enable (وغيرها) في TMenuItem objects لإنشاء قوائم ديناميكية .

• استخدم خاصية الـ Items في الـ MainMenu objects ، الـ PopupMenu ، والـ MenuItem التي تحتويها كلاً من الأنواع الثلاثة للـ menu objects . قم بتحديد قيم فهرس الصف بالمدى من صفر إلى Count - 1 . تعتبر Items من الـ TMenuItem ، والتي لها خاصية الـ array ؛ وبذلك ، يمكنك استدعاء method ، مثل Insert for Items ، أو يمكنك استخدامها كـ array .

• استخدم standard accelerator shortcut keys عند الإمكان . يمكنك تعيين أي مجموعة مفتاح مثل accelerator shortcut . وللحصول على الأداء الأمثل لوظائف ShortCut ، الـ ShortCutToKey ، الـ TextToShortCut ، والـ ShortCutToText الموضحة في هذا الباب .

• إن أسهل طريقة لإنشاء قوائم ديناميكية هي إضافة MainMenu objects متعددة على الـ form وتعينها في وقت التشغيل لخاصية الـ Menu . يمكنك أيضاً تعيين PopupMenu objects لخاصية الـ PopupMenu الخاصة بالـ form .

• وهناك أسلوباً آخر سهلاً لإنشاء قوائم ديناميكية وهو تحديد خاصية الـ Visible في الـ MenuItem بـ True و False . وهذا يجعل بنود القائمة- أو حتى القوائم بأكملها على menu bar- تظهر وتختفي بموجب تحكم البرامج .

• استخدم الـ Add method ، Delete ، Insert ، و Remove لتعديل عناصر القائمة في وقت التشغيل . على سبيل المثال ، يمكنك استخدام هذه الـ procedures لإدخال اسماء ملفات في أسفل قائمة الـ File . إنني أجد من السهل إنشاء بنود قائمة وهمية لهذا الغرض وتحديد خصائص الـ Visible لها بـ True عندما يفتح المستخدمين ملفات جديدة .

• يمكنك إنشاء TMenuItem objects جديدة في وقت التشغيل . استدع الـ Create لإنشاء كل object ، واستخدم Add أو Insert لإدخال العنصر في القائمة . قم بتعيين الـ OnClick event handler الخاص بالبند .



- أدمج MainMenu objects متعددة بتحديد قيم الـ MenuItem GroupIndex وخصائص الـ AutoMerge كما هو موضح فى هذا الباب . يمكنك دمج MainMenu objects فى نفس الـ form، أو فى forms مختلفة . إن تطبيقات الـ MDI والـ OLE تستخدم تقنيات دمج قوائم مختلفة .

- يمكنك إضافة أوامر لقائمة نظام خاصة بتطبيق ما . إن الخطوات المذكورة فى هذا الباب توضح كيفية القيام بهذا، وتوضح أيضاً كيفية إنشاء الـevent handler لرسائل الـ Windows .

- استخدم menu templates، resource scripts لإنشاء portable menus لتشارك فيما بين تطبيقات متعددة . يمكنك أيضاً إنشاء الـ menu templates الخاصة بك .

- يضيف Delphi 4 الـ ActionList للـ Standard . استخدم الـ ActionList لإنشاء Action objects تملك events . وخصائص يمكن للـ components الأخرى مثل MenuItem و Button أن تتشارك فيها .

إن Delphi يجعل من تحكم الزر فى الـ Windows علماً، كما ستكتشف فى الباب القادم . بالطبع يمكنك إنشاء أزرار، check boxes، و radio buttons، ولكن يمكنك تنسيق واجهة تطبيق المستخدم الخاصة بتطبيقك وذلك باستخدام BitBtn component objects الجرافيكى و component objects الأخرى الخاصة بـ Delphi. وسوف نلقى الضوء أيضاً على النص الثابت والتحكم لأعلى وأسفل .



الباب السادس

Attaching Buttons and Check Boxes

محتويات هذا الباب:

- Components
- ازرار اساسية
- ازرار ملونة
- إغلاق نافذة
- مجموعات الازرار
- Spin buttons
- Static text
- Up-down button controls

تعتبر الازرار و check boxes هى لب واجهة تطبيق المستخدم الجرافيكية ، ويقدم Delphi خزانة أدوات من الـ push buttons ، radio buttons، check boxes، speed buttons ، bitmap buttons . يمكنك تجميع مجموعات من button objects، يمكنك استخدام organizational components مثل الـ bevels والـ panels لترتيب نافذة مزدحمة ممتلئة بالإزرار والـcontrols الأخرى.

فى هذا الباب ، إننى اوضح كيفية استخدام Delphi's button objects والـ components المرتبطة بها مثل GroupBox، RadioGroup ، SpinEdit ، و

SpinButton . إننى سوف اوضح ايضاً كيفية إنشاء واستخدام bitmap ملونة ، والتى تسمى glyphs ، والتى تظهر على الـ BitBtn والـ SpeedButton . يمكنك ايضاً عرض صور الـ glyphs الـ SpinButton . واخيراً، سوف اشرح static text controls (والتى تضيف خصائص مثل الـ borders الى standard label text) ، وكذلك الازرار العلوية والسفلية الخاصة بالـ Win32 ، والتى تشبه الـ SpinButton ولكنها متأصلة فى الـ Windows ذو الـ ٣٢ بت .

ملحوظة: يقدم هذا الباب الـ Panel والـ SpeedButton. انظر الباب التالى للحصول على مزيد من المعلومات حول استخدام هذه الـcomponents لإقامة toolbars و status panels .

Components :

ان الـ components المرتبطة بالازرار والخاصة بـ Delphi هى :

- **Bevel:** ويبدو هذا الـ visual component وكأنه فراغات مستطيلة فى النافذة. ويمكن للـBevel ايضاً ان يعرض خطوطاً راسية وافقية والمعروفة جيداً لمصممى واجهة التطبيق على انها dips ، speed bumps ، والتى تفيد فى تقسيم النافذة الى قطاعات . [Palette: Additional] .

- **BitBtn:** قد تسمى هذه الـ component بـ " push button ذو مهارة " . يعمل الـ BitBtn مثل Button component ولكن يمكنه عرض ايقونة ملونة ، تسمى glyph ، والتى تمثل من الناحية البصرية عمل الزر . [Palette: Additional] .

- **Button:** يعتبر Windows button control معيارى محتوى من خلال Delphi components . إننى سوف اشرح المزيد عن الـ Buttons فى هذا الباب ولكنك فى الغالب سوف تريد استخدام الـ Button بدلاً منه . [Palette: Standard] .

- **CheckBox:** وهذا ايضاً يعد Windows control معيارى، ويتكون من مجموعة من labels والـ boxes يأخذ قيم on و off . إنه مفيد الى درجة كبيرة فى تصميم dialog boxes مع مجموعة من خيارات البرنامج . [Palette: Standard] .



• **GroupBox**: وهو Windows control معياري آخر، وهذا الـ component يقوم بتجميع RadioButtons- objects متعددة و الـ objects الأخرى . يمكن للمستخدمين ضغط الـ Tab للتحرك بين الـ GroupBoxes المتعددة، ثم ضغط مفاتيح الاسهم للإختيار من الـ controls المتجمعة . والـ RadioGroup component، والذى يبسط عملية تطبيق الـ RadioButton . [Palette: Standard] .

• **Panel**: هذا الـ component يقدم الـ platform لتقسيم نافذة مزدحمة لإنشاء toolbars و status panels (سيتم شرحها والباب الثانى) . يمكن ان يظهر على انه سطحاً مرتفعاً، أو يمكن ان يبدو مفرغاً وبه حدود زوايا مائلة متنوعة ان الـ Panels لا يكون لها أى input focus، وتعمل كحاوية للـ controls . [Palette: Standard] .

• **RadioButton**: يمكنك إضافة الـ RadioButton متعددة فى الـ Panel أو الـ GroupBox، ولكن فى اغلب الحالات، يسهل استخدام الـ RadioGroup لإنشاء مجموعات الـ RadioButton . ادخل الـ strings فى خاصية الـ Items للـ RadioGroup لإنشاء ازرارك . [Palette: Standard] .

• **SpeedButton**: إنك تستخدم عادة الـ SpeedButtons لإنشاء toolbars، كما سيوضح الباب التالى . ولكن، يمكنك إضافة SpeedButtons منفردة على forms . ان تطبيق الـ Calc32 الخاص بهذا الباب- آلة حاسبة ذات ٣٢ بت متوفرة على القرص المدمج المرفق بهذا الكتاب- يستخدم الـ SpeedButton بهذه الطريقة . [Palette: Additional] .

• **SpinEdit**: موضوع على Samples VCL palette ويعتبر هذا الـ component من الـ custom component ولكنه مفيد فى حد ذاته . سوف اشرح فى هذا الباب ايضاً كيفية استخدام الـ SpinButton المرتبط به [Palette: Samples] .

• **StaticText**: استخدام هذا الـ component الجديد نسبياً لإنشاء static text labels ذات خصائص بصرية مثل sunken border لإنشاء نوافذ قطاعية،



فإن هذا الـcomponent يعتبر اسهل من تجميع الـ Label والـ Panel المتعددة . [Palette: Additional] .

- **UpDown:** هــذا الـcomponent يعتبـــر 32-bit control والذى يشبه الـ SpinButton control وهو موضح أيضاً فى هذا الباب . [Palette: Win32] .

ازرار اساسية:

يقـدم الـWindows ثلاث انواع قيـاسيـة من الازرار، والتى تحـتـويهـا الـ Delphi encapsulates . والازرار الثلاثة والـ components المقابلة لها هى :

- **push buttons:** Button component
- **Check boxes:** CheckBox component
- **Radio buttons:** RadioButton component

يتـعـامل انواع الـcomponents الثـلاث مع نفس الـ events، على سـبـيل المثال، يمكنك إدخال عبارات فى الـ OnClick event handler لأداء اعمال عندما يختـار المستـخـدمون زراً، radio button، check box ، للحصـول على تحكم اكبـر فى اختـيـار الزر، استـخـدم الـ OnMouseDown أو الـ OnKeyDown . يمكنك ايضاً برمجة اعمال لإطلاق الزر وذلك بإنشاء الـ OnMouseUp event .

اجـعل قبل أى رمـز فى خـاصـيـة الـ Caption للزر عـلامـة & للإشـارة الى Alt-shortcut accelerator key . على سـبـيل المثال، يمكن للمـسـتـخـدمين ان يضغطوا Alt+C لاختـيـار حرفاً فى خاصية الـ Caption . على سبيل المثال، يشير الـ Caption E&xit الى الـ Alt+X على انه button's shortcut key .

Push buttons :

ان Caption E&xit سهلة الاستخدام . فقط اضغط الـ Button object على الـ form، وقم بتعيين string لخاصية الـ Caption للـ button's label . كما رأيت فى الأمثلة العديدة الأخرى، لأداء اعمالاً عندما يضغط المستخدمون زراً ما، قم بإنشاء event handler لـ OnClick event الخاص بالزر، وادخل عبارات بين البداية والنهاية .

بالنسبة للزر الأفتراضى حسب النظام، قم بتغيير خاصية الـ Default الى True . يجب ان يكون زراً واحداً على الـ form ، (عادة الذى يحمل بطاقة OK) افتراضياً حسب النظام، والذى يمكن للمستخدمين اختياره، بضغط Enter . لربط زر (غالباً يكون Cancel) بمفتاح الـ Esc ، حدد خاصية الـ Cancel للزر بـ True . بغض النظر عن تحديدات الـ Cancel والـ Default ، يجب عليك إنشاء event handler للأزرار.

وتتأخذ الازرار والـ controls الأخرى الـ Font الخاص بالـ form . للحصول على عرض افضل، قم بتغيير الـ Font الخاص بالـ form ليكون TrueType font مثل Arial . ولكن، عند فعل هذا، تأكد من اختيار font متاحاً فى كل الـ Windows installations . ان الـ fonts المثالية المتوفرة فى Windows 95 installation مذكورة هنا. يمكنك اختيار fonts مختلفة للازرار الفردية والـ controls الأخرى.

Note

ملحوظة: تعتبر الـ TrueType fonts قابلة للتغير من ناحية ، الحجم ، وكذلك bitmap fonts تعتبر قابلة للتغير ، ولكن انظر جيداً فى احجام النقطة المذكورة الخاصة بهم.

- **الـ TrueTypeFonts:** Arial؛ Arial Bold ؛ Arial Bold Italic ؛ Arial Italic؛ Courier New ؛ Courier New Bold ؛ Courier New Bold Italic؛ Courier New Italic ؛ Symbol ؛ Times New Roman ؛ Times New Roman Bold ؛ Times New Roman Bold Italic ؛ WingDings.

- **Bitmap Fonts:** Courier 10، 12، 15، Modern ؛ MS Sans Serif 8، 10، 12، 14،، 18، 24، Roman ؛ Script ؛ Small Fonts ؛ Symbol 8، 10، 12، 14،، 18، 24 .

فى حالات نادرة قد تريد ان تجبر المستخدمين على اختيار push button بعينه، زر OK مثلاً. لا يمكنك ان تفعل هذا بتحديد خاصية الـ Default للزر بـ True لان المستخدمين مازال باستطاعتهم ان يضغطوا Tab لتحويل الـ Focus الى

آخر مثل Cancel . قد يكون هذا محيراً، خاصة للمستخدمين الجدد للحاسوب الذين، لم يكتسبوا خبرة، قد يفشلوا فى ملاحظة outline الذى يتركز حول الـ focused control .

وأحد الحلول لهذه المشكلة هو ابطال كل الازرار والـ objects الأخرى على الـ form بتحديد خصائص الـ Enabled بـ Fals . لفعل هذا قم بتعريف متغير الـ I من نوع Integer، واستخدم code كالآتى :

```
for I := 0 to ComponentCount - 1 do
  if Components[I] is TWinControl then
     TWinControl(Components[I]).Enabled := False;
```

وتتوصل الـ code السابقة الى Components array فى الـ form لتحديد خصائص الـ Enabled لكل الـ control objects بـ Fals . وتختبر عبارة if اذا ما كان TWinControl object. (وهى الـ class الأم التى تنحدر منها الـcontrol objects مثل الازرار) . اذا كان كذلك، يحدد البرنامج خاصية الـ Enabled للـ control بـ False بوضع الـ Components[I] فى الـ TWinControl . والوضع هنا أمر ضرورى لان محتويات Components array من النوع العام، TComponent، الذى ليس له خاصية Enabled .

بعد تنفيذ الـ code السابق، تصبح كل الـ controls على الـ form، لا يسمح بالتعامل معها. لإنهاء البرمجة، قم بتشغيل زراً واحداً وانقل للـ focus الى هذا الزر . يعتبر هذا هو الزر الوحيد الذى يمكن ان يضغطه المستخدمين :

```
OKButton.Enabled := True;  { Enable button and set the focus }
OKButton.SetFocus;         { so pressing Enter selects it. }
```

: Check boxes

تعتبر CheckBox components من ضمن ابسط ادوات تصميم واجهة التطبيق الـ Windows ولكن من اكثرها إفادة . قم بادخال الـ CheckBox object فى form، وحدد خاصية الـ Checked له بـ True لعرض علامة الاختيار صح فى المربع، أو بـFalse لعدم الحصول على علامة الصح . فى وقت التشغيل، يمكنك اكتشاف اذا ما كان الـ CheckBox عاملاً أو مغلقاً باستخدام الـ code، كما يلى :

```
if MyCheckBox.Checked
  then {do something};
```

يعتبر الـ CheckBox فى الغالب on-and-off toggle ولكن يمكن ان يكون ايضاً مفتاح ثلاثى مفتوحاً، أو لا تستطيع التعامل معه أو مغلقاً. ان الـ CheckBox الذى لا يتعامل معه يكون له علامة صح رمادية فى مربعه.

لانشاء CheckBox ثلاثى، حدد خاصية الـ AllowGrayed له بـ True، ثم حدد State بواحدة من ثلاث قيم: cbChecked، أو cbGrayed، أو chUnchecked يمكن للمستخدمين الآن اختيار الـ CheckBox لفتحه أو إغلاقه، أو إيقاف التعامل معه.

اذا كانت الـ AllowGrayed محددة بـ cbGrayed، و Fale بـ State، تظهر علامة صح لإيقاف التعامل فى CheckBox control. يمكن للمستخدمين اختيار الـ control لاغلاقه أو فتحه، لكن لا يمكنهم إعادة CheckBoxes الى حالته من إيقاف التعامل معه.

Tip

فكرة: يمكنك تلوين خلفية الـ CheckBox باختيار قيمة الخاصية الـ Color. على سبيل المثال، اجعل الـ Color بـ clBtnShadow لتلوين الـ CheckBox بنفس لون تحديد النظام لأظلال الزر. أو، اختر لوناً ثابتاً مثل الـ clOlive من قائمة اللائحة لخاصية لـ Color.

لا يمكنك تلوين الـ Button كما تفعل فى الـ CheckBox. وهذا يعتبر حد من حدود الـ Windows. للحصول على مزيداً من الازرار الملونة، استخدم الـ BitBtn بدلاً من الـ Button component.

فى كثير من الحالات، تقوم بإنشاء CheckBoxes متعددة. إننى اجده من السهل ان تفعل هذا اولاً بادخال CheckBox object واحد، أضغط Ctrl+C لنسخة فى clipboard، ثم اضغط Ctrl+V تكراراً لإضافة objects جديدة على الـ form (قد تختلف هذه المفاتيح إعتماداً على الـ Environment Options الخاصة بك- افتح قائمة Edit للفحص). ثم اقوم باختيار كل objects، غير خصائص الـ Name والـ Caption له، وانقل كل object الى موضعه النهائى.

:Radio buttons

والـ Windows controls المعـيـاري الثـالث والأخـيـر وهو الـ RadioButton، والذي يحـتـويه الـ Delphi في الـ RadioButton component. تشبه الـ RadioButtons شرائح البطـاطس - فواحـدة منهـا لا تكفي. إنك دائماً تستخـدم أثنان من الـ RadioButton على الأقـل (يعتبر الـ CheckBox اكثر ملائمة من الـ RadioButton للتحكم في الاغلاق والفتح الفردي). يمكنك تلوين الـ RadioButton كما تفعل في الـ CheckBoxes.

عندما تحتوي form على RadioButton متعددة، يمكن للمستخدمين ربما زراً واحداً فقط في المرة فتحاً أو إغلاقاً. ولكن عندما تدخل مجموعات متعددة من الـ RadioButton، فإنها تعمل وكأنها مجموعة فردية واحدة، وهذا مالا تريد غالباً. والحل هو تجميع كل مجموعة من الـcontrols بحيث يستطيع المستخدمين اختيار زراً واحداً من كل مجموعة، واضغط Tab لتحويل الـ focus من إحدى المجموعات الى الأخرى (والى controls اخرى). سوف اشـرح المزيد عن تجميع الـ control objects في فصـل "مـجـمـوعـات الازرار"، مـؤخـراً في هذا الباب. وكذلك، انظر "استخدام الـ Radio Groups" لمعرفة تقنية بديلة لتجميع الـ RadioButton.

الـ RadioButton تقـدم double-click event وهو غـيـر مـتـاح للـ RadioButton أو check boxes. اضـغـط مـرتين الـ OnDblClick في الـ Object Inspector لإنشـاء بالـ event handler. يمكن للمستخدمين بعد ذلك ان يضغطوا مـرة واحدة الـ radio button control أو يمكنهم ان يضغطوا مرتين الـ object لأداء اعمالاً مختلفة.

ازرار ملونة:

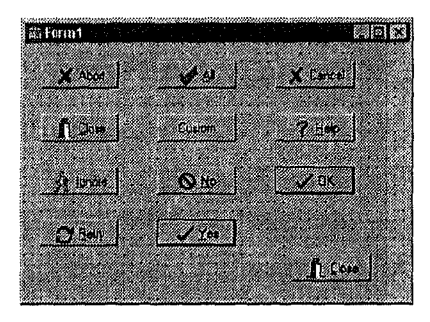
للحصـول على عرض أخـاذ، استخدم الـ BitBtn object بدلاً من الـ Windows Button المعياري. ان الـ BitBtn يعمل مثل push button، ولكن يمكنه عرض. bitmap ملونة، تسمى glyph، والتي تذكر المستخدمين بمل يفعله الزر. ان النظرات قد تكون خادعة، ولكن القليل من التجميل له تأثير بعيد، والـ glyphs يمكن ان تساعد على تزيين نافذة تكون رمادية مملة بغير هذه الـ glyphs.

يمكنك ايضاً الاختيار من بين الـ BitBtn المعرفة سابقاً العديد مع الـ glyphs الأفتراضية التى تمثل عمليات معيارية الـ Yes ، الـ No ، الـ Close ، الـ Help الـ Ignore أو الـ Retry .

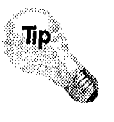
يمكن ايضاً ان يوفر الـ Delphi الـ SpeedButton والذى يشبه الـ glyph bitmap مع SpeedButton وغالباً بلا Caption . وكما سيوضح الباب القادم ، يمكنك استخدام الـ Panels و الـ SpeedButton لإنشاء toolbars ، ولكن يمكنك ايضاً إضافته SpeedButton مستقلة فى نافذة الـ form . ويظهر تطبيق هذا الباب Calc32 - آلة حاسبة اعداد صحيحة لواضعى البرامج ذات ٣٢ بت - كيفية استخدام الـ SpeedButton كـ controls فردية .

ازرار الـBitmap:

يعتبر الـ BitBtn الأول فى Additional VCL palette . ان اسهل طريقة لإستخدام هذا الـ component هى اضافتة على الـ form ، ثم تحديد خاصية الـ Kind له بواحدة من القيم الموضحة فى شكل (٦-١) على سبيل المثال ، bkHelp . حدد الـKind بـ bkCustom لعرض صورة الـ glyph الخاصة بك أو لا يعرض شيئاً . اذا لم تعرض صورة glyph على الـ BitBtn رغم ذلك ، يمكنك استخدام الـ Button .



شكل (٦-١): form يحتوى على نوع من الـ BitBtn توضح الـ objects القيم المدخلة فى خصائص الـ Kind للـ object



فكرة: اذا حددت خاصية الـ Kind للـ BitBtn بـ bkClose ، يمكن للمستخدمين اختيار الزر لإغلاق نافذة الـ form الخاصة به . ويقوم الزر بهذا عن طريق التحديد الداخلى للـ ModalResult بقيمة مثل الـ

mrOk . لا يجب عليك كتابة الـ code الخاصة بك لتفعـــل هـــذا . ولكن ، اذا لم تكن تريـــد للـ Close BitBtn أن يغلق النـــافذة ، أولاً حـــدد الـ Kind بـ bkClose ثم غـيـره الى bkCustom . تبـقى الـ button label والـ glyph الخـــاص به كـمــا همـا ، ولكن يمكنـــك الآن تعـيين الـ event handler الخاص بك للـ OnClick الخاص بالـ object .

يمكنك تعيين صورة للـ BitBtn . اضـغط مرتين قيمة خاصية الـ Glyph افتح الـ Picture Editor . اختر Load وحدد الـ bitmap . يمكنك العثور على مجموعة من الملفــات فى دليل Images\Buttons الخاص بـ Delphi . بعد تحميل bitmap ، اختر Save لنسخ الصورة فى ملف جديد . اختر OK لنسخ صور من الـ bitmaps فى الـ BitBtn . وهذه الخطوات تنفـــذ أيضـــاً مع الـ SpeedButtons .

لا يجـب عليـك توفـيـر ملف bitmap مع ملـف الـ code الخـاص بتطبيقك . ولكن ، قد تريد نسخ الـ bitmap والملفات الأخرى الخاصة بـ Delphi فى أدلة التطوير الخاصة بك حتى تنشئ مجموعة كاملة من الـ source files لتطبيقك .

لتعديل موضع الـ glyph الخاص بالـ BitBtn بالنسبة للـ caption التابع له ، حـــدد خـاصـيـــة الـ Layout بـ blGlyphBottom ، أو blGlyphLeft ، أو blGlyphRight ، أو blGlyphTop . قم بتـجـربة هذا على الـ BitBtn لتـرى تأثيرها . للحصول على مظهر معيارى . استخدم القيمة الإفتراضية blGlyphLeft ، والتى تضع الـ glyph الى اليسار من الـ caption الخاص بالزر . وكذلك قم بتعديل خاصية الـ Margin لفصل الصور عن حدود الزر- صفر (أو 0) لعدم الفصل ، 1 لمسافة pixel واحد ، أو two pixels ، وما الى ذلك . لوضع الـ glyph فى المركز ، حدد الـ Margin بـ 1- .

لمزيد من المعلومات حول glyph bitmaps ، أنظر "المزيد عن الـ Glyphs" بعد الفصل التالى حول الـ SpeedButtons .

تعديل نمط عرض الـ BitBtn باستخدام خاصية الـ Style

إن النسخ الأولى من Delphi كانت تسمح بتعديل نمط عرض الـ BitBtn باستخدام خاصية Style. للمرجع، لقد ذكرت أوصاف تحديد هذه الخاصية، ولكنها مهجورة الآن، بالرغم من أنه قد يكون لها تأثير بصرى فى الـ Win32 والـ Windows 3.1 (القيمة الإفتراضية للـ BitBtn من bsAutoDetect):

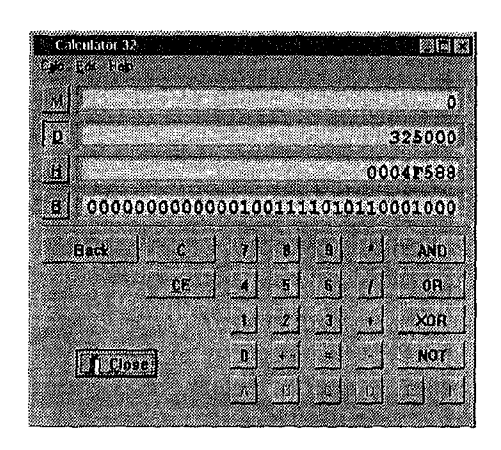
- bsAutoDetect: تستخدم نمط الـ bsNew للـ Windows 95 ونمط الـ bsWin31 للـ Windows 3.1.
- bsNew: تستخدم دائما نمط زر الـ Windows 95 الجديد، مع تأثيرات ظلال أكثر رفعا على الحواف السفلية واليمنى.
- bsWin31: دائما يستخدم نمط زر الـ Windows 3.1 مع ظلال أكثر كثافة على الحواف اليمنى والسفلية.

:Speed buttons

تظهر الـ SpeedButton غالباً على toolbar، ولكن يمكنك أيضاً إنشاء SpeedButton منفردة كما سأشرح هنا. يمكن أن تعرض الـ SpeedButton أيضاً أيقونات ملونة تسمى glyphs. أنظر الباب التالى لمزيد من المعلومات عن الـ SpeedButton والـ Panels لإنشاء toolbars.

بشكل عام، إنك تبرمج احداثى الـ SpeedButton كما تفعل فى الـ push buttons، ولكن الـ SpeedButtons تتعرف على خمسة events فقط. استخدم الـ OnClick والـ OnDblClick لأداء أعمال للضغط على الفأرة الفردية والمزدوجة. استخدم OnMouseDown و OnMouseMove و OnMouseUp للحصول على تحكم أدق فى إختيار الزر- على سبيل المثال، استكشاف Shift parameter للـ OnMouseDown ليحدد ما اذا كان هناك زراً مضغوطاً. (أنظر الباب الرابع لمزيد من المعلومات عن نوع بيانات الـ TShiftState الخاص بهذا الـ parameter).

على القـرص المدمج: إن الميزة الأساسية فى الـ SpeedButton عن أنواع الأزرار الأخرى هى قدرته على العمل بطريقتين : on/off toggle أو "sticky" button ، ليبقى الى اسفل عندما تدفعه وهذه السمات تجعل الـ SpeedButtons الاختيار الامثل فى دليل الـ Calc32 . يوضح شكل (٦-٢) ظهور الـ Calc32 . ان به SpeedButton objects يمكن ان تعمل on/off toggles (مثل digit entry keys) و "sticky" button تبقى الى اسفل عندما تدفعها (مثل ازرار الـ Memory ، الـ Decimal ، الـ Hexadecimal ، والـ Binary فى عنصر قائمة الـ Calc .



شكل (٦-٢): لإنشاء آلة حاسبة قمت باستخدام الـ SpeedButton التى تعمل كـ on/off toggle و "sticky" button وتبقى الى اسفل عندما تضغطها

والسبب الثانى لاستخدام الـ SpeedButtons يتعلق بالـ resource . لأن SpeedButton ليس لهـا window handles و associated structures internal فى الـ Windows لذا فإن الـ form التى بها دستة من الـ SpeedButtons تستخدم ذاكرة اقل من دستة من الازرار المعيارية . وتعرض الـ SpeedButtons ايضاً بصورة اسرع من الـ control المعيارية .

يعتبر الـ Calc32 آلة حاسبة للاعداد الصحيحة ذات ٣٢ بت ، مناسبة لمهام البرمجة مثل تشكيل انماط البت والتحويل بين الكسور السداسية ، والثنائية والعشرية يعتبر الـ Calc32 بسيط فى الاستخدام اضغط ازرار الـ D أو الـ H أو الـ B لاختيار ادخال عشرى ، سداسى أو ثنائى . ادخل قيم باستخدام لوحة المفاتيح أو بضغط

ازرار يظهر دائماً حقول العرض بالقرب من قمة النافذة قيماً فى ثلاث جذور بغض النظر عن نمط الادخال. تكون القيم المخزنة فى حقل الذاكرة (M) دائماً عشرية. لا يقدم الـ Calc32 قيم من نوع floating-point .

على القرص المدمج: ان الـ source code للـ Calc32 طويلة جداً على ان تذكر هنا، بالطبع، جميع الملفات موجودة على القرص المدمج. إننى اشير الى فصول من الـ code من وقت الى آخر. لاختيار التطبيق، افتح ملف مشروع الـCalc32.dpr فى دليل الـ Calc32 .

يوضح الـ Calc32 ان الـ SpeedButtons ليس عليها ان تعرض صور الـ glyph حتى تكون مفيدة. لإنشاء ازرار برنامج، ادخلت خصائص الـ Caption مثل 1، 2 ، 3 ، C ، CE ، وحددت الـ Glyph بـ None (القيمة الافتراضية).

لقد اخترت الـ SpeedButtons للـ Calc32 بدلاً الـ Button المعيارية لان اثنين أو أكثر من الـ SpeedButtons يمكن ان تعمل مثل مجموعة من الـ RadioButtons . لعمل هذا، قم بتعيين قيم الـ GroupIndex لتحديد مجموعات الـ SpeedButtons اذا كانت الـ GroupIndex تساوى 0 (أى صفر) "وهو القيمة الافتراضية" ، يعمل الـ SpeedButtons مثل spring-loaded on/off switch. وتنشئ قيم الـ GroupIndex الموجبة الأخرى مجموعات من الـ SpeedButtons . على سبيل المثال، حدد سلسلة من الـ GroupIndexes الخاصة بالـ SpeedButtons بـ 3. يمكن للمستخدمين ان يختاروا SpeedButtons واحد فقط فى هذه المجموعة، واختيار احد الازرار يغلق تلقائياً الـ control الذى تم اختياره حالياً.

فى الـ Calc32، يكون للـ SpeedButtons الذاكرة (M)، العشرية (D) قيم GroupIndex تساوى 1 . ان باختيار احد هذه الازرار يظهر الآخرين. يبقى الزر الذى تم اختياره الى اسفل حتى تختار آخر فى المجموعة.

ان احدى المشكلات التى واجهتنى وانا اكتب الـ Calc32 كانت تعيين event handlers لمجموعات الـ SpeedButtons . لقد اردت ان استخدم اسماء الـ procedure الخاصة بى بدلاً من تلك التى يعينها Delphi حسب النظام الافتراضى للبرنامج. لفعل هذا، قمت بأداء الخطوات التالية :

١- اولاً، اختـرت object من المجـمـوعـة- عـلى سـبـيل المثـال زر الـ A السداسى.

٢- ضغطت مرتين الـobject لإنشاء الـ ButtonAClick طبقاً لنظام البديل الافتراضى للبرنامج.

٣- بعـد ذلك، اسـتـخـدمت امـر الـ Search|Replace التـابع لـ Delphi لإعـادة تسـمـيـة كل حـدوث للـButtonAClick بـ DigitButtonClick، والذى يصف بدقة أكثر ما يفعله الـ procedure.

٤- واخيراً، اخترت كل الازرار فى المجموعة، بما فى ذلك الزر الاصلى من الخطوة الأولى. باستخدامEvents page tab الخاص بالـ Object Inspector، قمت بتحديد الـOnClick المشترك بـ DigitButtonClick.

:Glyphs

الـ glyph هى Windows bitmap معروضـة على شكل SpeedButtons أو BitBtn. بالرغم من انه لا يوجد حـدود لحجم الـ glyph، فإن الصورة المعيارية ١٦×١٦ pixel، فى الـ component، تعتبر خـاصية الـ Glyph، object من نوع الـ TBitmap.

كم هو مـوضح فى شكل (٦-٣)، قـد تحـوى الـ glyphsمن واحـد الى اربع صور منفصلة، مخزنة جنباً الى جنب فى ملف Bitmap، كلاً منهما يجب ان يكون متساوياً فى العـرض والطول (غالباً ١٦×١٦ pixel) يعرض Delphi كل صورة لتمثل حالة زر مختلفة:

١- Up (العرض الطبيعى).

٢- Disabled (معتم؛ Enabled = False).

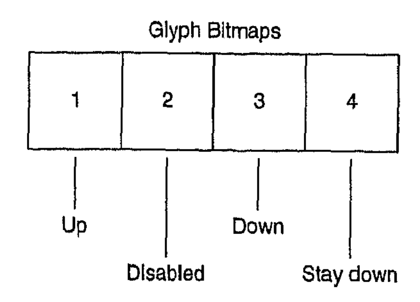
٣- Down (محولة ومن المحتمل معتمة).

٤- Sty down (للـ SpeedButtons فقط).

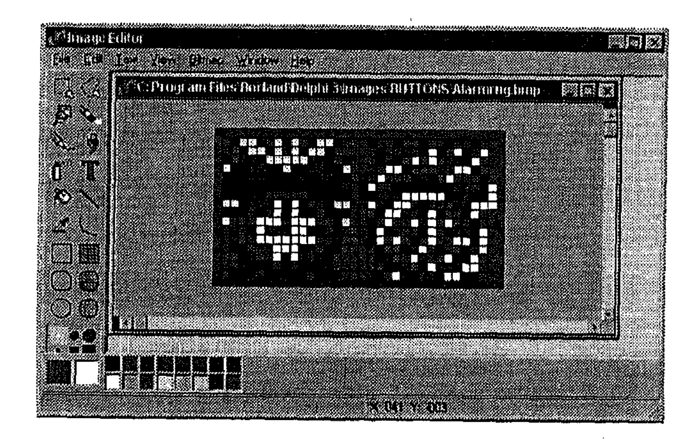
لعرض زر فى حالته الطبيعية، يستخدم Delphi الصورة الأولى. لعرض زر فى حالة دفعه، يعرض Delphi الصورة الثالثة للـglyph. اذا لم يكن هناك صورة ثالثة لـ glyph، ينقل Delphi بدلاً من ذلك bitmap الى اسـفل والى اليـمين.



يوضح شكل (٦-٤) المناظر الموسعة والطبيعية للملف الـ Alarmrng.bmp المتاح مع Delphi . لان bitmap يفتقد صور ثالثة ورابعة ، ينقل Delphi البت الخاصة بالـ glyph الى اسفل ويميناً لتمثل حالات الزر المدفوع . يمكنك توفير صور ثالثة ورابعة لإحياء الـ BitBtn أو SpeedButton عند اختياره .



شكل (٦-٣): تحتوى الـ Glyph bitmaps من واحد الى اربع صور جنباً الى جنب، كلاً منها له هدف مختلف. على سبيل المثال، يستخدم Delphi الصورة الثانية لعرض زر الابطال



شكل (٦-٤): ملف الـ Alarmrng.bmp يحتوى على glyph من جزئين. يستخدم Delphi الصورة الأولى لحالة الزر الطبيعية وهى انه غير مدفوع، والصورة الثانية للازرار التى تم ابطالها

تحميل وتعيين glyph bitmaps:

عند تحميل glyph bitmap ، حدد خاصية الـ NumGlyphs بعدد الصور التى يحتويها . وهذا يجب ان يكون قيمة فيما بين واحد واربعة . فى العادى ، يعطى Delphi القيمة الصحيحة من حجم الـ bitmap . على سبيل المثال ، يفترض

Delphi ان الـ bitmap الذى له ٣٢ بت عرضاً ×١٦ بت طولاً يحتوى على اثنين من الـ glyph ذات ١٦×١٦ .

يمكنك تعيين صور الـ glyphs عندما تصمم form ، أو يمكنك تحميل ملفات الـ bitmap فى وقت التشغيل ربما لإحياء الـ BitBtn أو SpeedButton object ، لتغيير مظاهره المؤسس على ظروف خارجية . لتحميل الـ glyph فى وقت التشغيل ، اضف BitBtn object على الـ form ، اضغطاً مرتين ، وادخل هذه العبارات OnClick (قم بتعديل اسم المسار طبقاً للمكان الذى قمت فيه بتركيب الـ Delphi) :

```
BitBtn1.Glyph.LoadFromFile(
  'c:\delphi\images\buttons\alarm.bmp'); { Or another file }
BitBtn1.NumGlyphs := 2;            { Important step! }
```

تستدعى العبارة الأولى LoadFromFile procedure للـ Glyph object لتحميل ملف الـ Alarm.bmp المقدم مع Delphi تعدد العبارات الثانية عدد الـ glyphs بإثنين.

Note

ملحوظة: دائماً حدد الـ NumGlyphs بالقيمة السليمة عند تحميل glyph bitmaps فى وقت التشغيل . يحسب Delphi تلقائياً عدد الـ glyphs فى ملف bitmap فى وقت التصميم فقط .

يمكنك ايضاً تعيين أى TBitMap object لخاصية الـ Glyph للـ SpeedButton أو الـ BitBtn. ولكن ، بمن المهم ان تفهم ان Delphi يصنع نسخة من الـ bitmap لإدخالها فى شكل الزر . إنها مسؤليتك ان تحذف من الذاكرة أى TBitMap object افتراضية كنت قد أنشأتها . عند تحميل bitmap من ملفات قرص وتعيين bitmap الناتجة الى خاصية الـGlyph مباشرة ، لا يجب عليك حذف أى شئ لان الصورة منسوخة مباشرة فى الـobject . ولكن افترض انك حددت TBitMap object منفصل ، ربما لأنك تريد ان تستخدمه لأكثر من غرض . أولاً ، قم بتعريف الـ TBitMap object ، المشار إليه هنا بـ MyImage :

```
var
  MyImage: TBitMap;
```

قم بإنشـاء bitmap object فى الذاكـرة باسـتـدعـاء الـ :Create method الخاص بـ TBitMap class's :

```
MyImage := TBitMap.Create;  { Create MyImage object }
```

لديك الآن object فى ذاكـرة، يشـار إليه بـ MyImage، والذى يمكنك ان تحمل فيه ملف bitmap . يمكنك عندئذ تعيين الـ object لخاصية الـ Glyph الخاصة بالـ BitBtn أو الـ SpeedButton . يمكنك ايضاً استـخدام MyImage لأغراض أخرى . ها هى باقى الـ code :

```
MyImage.LoadFromFile
('c:\delphi\images\buttons\alarm.bmp');
BitBtn1.Glyph := MyImage;
{ ... insert other uses for MyImage here }
MyImage.Free;
```

تستـدعى العبـارة الأولى الـ LoadFromFile للـ MyImage . تنسخ العبارة الثانية الـ object فى خاصية الـ Glyph للـ BitBtn . إدخل استـخدمات اخرى للـ MyImage حيث يشير التعليق. عندما تنتهى من استخدام الـ TBitmap object، حـرر الذاكرة الخـاصـة به باستدعـاء free . هذه الخطوة الأخيـرة تعتبر اساسية؛ ان نسيت ان تحرر الـ TBitmap الذى انشأته، تبقى اجزاء منه فى الذاكرة حتى بعـد انتـهاء البـرنامج . يقـوم تطبيق الـ Delphi بتـحـرير أية objects يملكها الـ parent object تلقائياً؛ ولكن يجب ان يحدد البرنامج أية structure اخرى مثل bitmap pixels المتـعلقـة بالـ object . اذا لم يقم البـرنامج بتـحـرير مثل هذه الـ structure ، فـإن الطريقـة الوحـيـدة لاسـتـعـادة الذاكـرة المفـقـودة هى الخـروج من التطبيق. وهـذا لان الـ Windows ذو الـ ٣٢ بت يخـصص resource على اسس سـابقـة الإنتـاج . عندمـا ينتـهى الانتـاج (على سبيل المثـال، يخرج المسـتـخـدم من التطبيق)، يتم استعادة أى resource ناقصة تلقائياً . ولكن فى الـ Windows ١٦ بت، الطريقـة الوحـيـدة لاسـتـعـادة الـresource مـفـقـودة هى الخروج وإعـادة بدء الـWindows .

ان لون خلفـيـة الـ glyph يسـاوى لـون الـ pixel الواحـد فى الركن الايسـر السفلى . وكل الـ pixel الأخرى فى الـ glyph التى لها نفس اللون تعتبر شفـافة-

بمعنى آخر، عندما يختار المستخدم لون الخلفية الازرار يحل محل الخلفية لـ the glyph's. اذا قام المستخدمون بتلوين الازرار الخاصة بهم باللون الاحمر الخاص بعربات المطافى، فسوف تكون خلفيات الـ glyphs الخاصة بك مساوية فى اللون الاحمر، ولكن على الأقل سوف تظهر كالعائمة على سطح الزر. ولكن للأسف أى pixels حمراء ثابتة سوف تختلط باللون الاحمر الخاص بالزر. والطريقة الأكثر فعالية للتصدى لهذه المشكلة هى استخدام الواناً مختلفة قدر المستطاع فى صور الـ glyph الخاصة بك. بهذه الطريقة سوف تبدو الـ glyphs الخاصة بك صحيحة حتى للمستخدمين الذين يفضلون تخطيط الوان الحاسب المكتبى "Hotdog Stand" التابع لـ Windows.

ملحوظة: فى العادى، يكون لون (الخلفية) الشفاف للـ glyphs طبقاً للنظام الافتراضى للبرنامج هو لون الـ pixel الأول فى بيانات bitmaps فى الجانب الايسر السفلى. يمكنك استخدام خصائص الـ TBitmap TransparentColor والـ TransparentMode للتغلب على هذه السمات الافتراضية حسب النظام، على سبيل المثال، قم بتعيين قيمة TColor مثل الـ clAqua للـ TransparentColor. يغير هذا تلقائياً الـ TransparentMode الى tmFixed. للعودة الى استخدام pixel الركن الايسر السفلى كقيمة اللون الشفاف، حدد الـ TransparentMode مرة اخرى بقيمته الأفتراضية، هى tmAuto.

كيفية تحريك الـglyph:

يمكنك تحريك glyph bitmaps لعرض صور مختلفة عندما يضغط المستخدمون، مثلاً الـ BitBtn. اتبع الخطوات التالية لتجربة هذه التقنية:

١- اضف الـ BitBtn على الـ form لقد استخدمت الاسم الافتراضى حسب النظام هنا وهو BitBtn1.

٢- اختر خاصية الـ Glyph للـ BitBtn1، اضغط الزر البيضوى وقم بتحميل ملف Dooropen.bmp من دليل الـ Images\Buttons الخاص بـ Delphi.

٣- اضف التعريف التالى فى public section للـ TForm1 class (يمكن وضعهم فى الـ private section ايضاً) :

```
DoorShutBmp:  TBitmap;
DoorOpenBmp: TBitmap;
```

٤- قم بإنشاء الـ bitmap objects وقم بتحميل الصور فيها بإضافة هذه العبارات الى لـ OnCreate event الخاص بالـ from (انسخ ملفى الـ bitmaps، فى الدليل الحالى من دليل Images\Buttons الخاص بـ Delphi) :

```
DoorShutBmp := TBitmap.Create;
DoorOpenBmp := TBitmap.Create;
DoorShutBmp.LoadFromFile('doorshut.bmp');
DoorOpenBmp.LoadFromFile('dooropen.bmp');
```

٥- حرر الصور بإضافة، هذه العبارات الى الـ OnDestroy الخاص بالـ form :

```
DoorShutBmp.Free;
DoorOpenBmp.Free;
```

٦- قم بإنشاء الـ OnMouseDown event الخاص بالـ BitBtn1 (تأكد من اختيار الزر وليس الـ form. اضف هذه العبارة الى الـ procedure :

```
BitBtn1.Glyph := DoorShutBmp;
```

٧- قم بإنشاء OnMouseUp event, الخاص بالـ BitBtn1 ، واضف هذه العبارة الى الـ procedure :

```
BitBtn1.Glyph := DoorOpenBmp;
```

٨- اضغط F9 لتشغيل البرنامج. عندما تضغط الـ BitBtn، يغلق الباب. عندما تطلق زر الفأرة يفتح الباب.

وطريقة أخرى بتحريك الـglyph هى بإنشاء bitmap واحدة تحتوى على صور منفصلة لحالات الزر العليا السفلى. على سبيل المثال، يمكنك جمع ملفات الـ bitmaps : Dooropen.bmp و Doorshut.bmp bitmap، وتعيين الـ glyph

المكون من اربعة اجزاء لـ BitBtn . بهذه الطريقة تزيل الحاجة الى تحميل وتحرير bitmap objects المنفردة فى وقت التشغيل .

مجموعات الازرار:

فى النوافذ المزدحمة- configuration dialog على سبيل المثال- يعتبر من الصواب تصنيف الازرار المتعددة فى مجموعات . قم بترتيب مجموعات الازرار بثلاث طرق عامة :

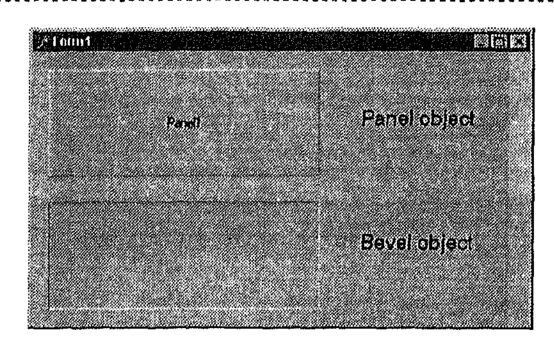
- استخدام الـ Bevel والـ Panel لإنشاء sections فى الـ form تظهر ذات فراغ أو مرفوعة . ادخل ازراراً فى هذه الـ objects .
- ادخل ازراراً فى الـ GroupBox .
- قم بإنشاء RadioGroup ، والذى يمكنه تلقائياً توليد RadioButtons متعددة . ببساطة أضف button labels فى Items array الخاص بالـ RadioGroup (المزيد عن هذا فيما بعد) .

استخدام الـ Bevel والـ Panel :

الـ Bevel والـ Panel ، تساعدك فى إنتاج عرض جذاب . ان الـ Bevel موجود على الـ Additional VCL ، والـ Panel على الـ Standard . إسقط هذه الـ objects على الـ form لتجربتها .

يمكن ان تستجيب Panel objects لـ events مثل الـ OnClick والـ OnMouseDown . وتشبة الـ Panels الازرار البسيطة التى لا تظهر مدفوعة عندما تضغطها . الـ Bevel تشبه الـ Panel ، ولكن لا تتعرف على أية events . وتعتبر الـ Panels مفيدة إيضاً فى إنشاء toolbars و status panels كما سيوضح الباب القادم . والـ Panels لها window handles ، ولذلك ، فإنها تستخدم system resources اكثر من الـ Panels . ولكن ، يمكن للـ Panels ان تحتوى controls اخرى مثل الـ Bevels ، وتعتبر الـ Bevels بصرية ، ولا نها تفتقد الـ window handles ، لا يمكن للـ Bevels ان تحتوى على Buttons أو controls أخرى .

شكل (٦-٥) يوضح الـ form الافتراضية مع الـ Panel و الـ Bevel . للـ Panel تعليق افتراضى (وهو Panel فى الشكل) ، ولكن يمكنك حذف قيمة خاصية الـ Caption لعرض سطح خالى .



شكل (٦-٥): form افتراضية مع Panel objects و Bevel، والتى تفيد فى ترتيب عرض مزدحم الى مقاطع بصرية

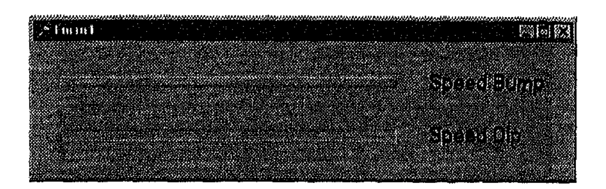
قم بتعديل خصائص الـ BevelOuter، الـ BevelWidt الـ BorderStyle، والـBorderWidt لمظاهر الـ Panel المختلفة. قم بتعديل خاصية الـ Shape لتغيير مظاهر الـ Bevel. (وقد يبدو محيراً ان الـ Panels تستخدم خصائصBevelxxx لتحديد مظاهرها، ولكن هذه تسمية ليس لها مدلول. ان Panels و الـ Bevels انواع مختلفة من الـ components).

لا يجب ان تتخذ الـ Bevel اشكال مربعات. يمكنك ايضاً استخدامها لإنشاء خطوط مقسمة، التى يسميها بعض واضعى البرامج speed dips وspeed bumps. على سبيل المثال حدد خاصية الـ Shape الـ Bevel بـ bsBottomLine لعرض speed dip افقى، والذى يفيد فى إنحناء نافذة لتكون قطاعات علوية وسفلية. استخدم bsLeftLine أو bsRightLine لإنشاء حواجز رأسية. قم بتعيين الـ Style ليصبح bsRaisedلإنشاء speed bump. يوضح الشكل (٦-٦) نماذج لـ speed dips وspeed bumps لكى اجعل هذا. مرئياً بشكل واضح هنا، فقد حددت. خصائص الـ Shape لأثنان من الـ Bevel بـ bsBox؛ ولكن يمكنك ان تحددها بـ bsBottomLine، أو bsFrame، أو bsLeftLine، أو بـ bsRightLine، أو bsTopLine لإنشاء speed dips وspeed bumps على خط واحد.

استخدام الـ GroupBoxes:

الـ GroupBoxes، يحتويه الـ Delphi فى component. استخدم الـ GroupBoxes لإنشاء مجموعة من الـ check boxes، radio buttons،

و other controls اخرى . يمكن للمستخدمين ان يضغطوا Tab لينتقلوا من أحد . الـ GroupBoxes الى آخـر ، ويمكنهم ضـغط مـفـاتيح السـهـم وقـضـيب المسـافـة للاختيار من بين الـ controls المجتمعة .



شكل (٦-٦): form حسب النظام الافتراضى وبها speed bump و speed dip

وتقيد الـ GroupBoxes الى حد كبير وترتيب radio button فى مجموعة منطقية . على سبيل المثال ، يمكنك إضافة radio button ومتعددة فى كل واحد من الأثنان من الـ GroupBox . يمكن للمستخدمين عندئذ اختيار احد الازرار من أى من المجموعتين اذا لم تضف الـ radio فى الـ GroupBoxes ، يمكن للمستخدمين ، ان يختاروا احد الازرار فقط فى النافذة .

يوضح الشكل (٦-٧) form بها إثنين من الـ GroupBoxes ، فى كل منهما ثلاث ازرار .

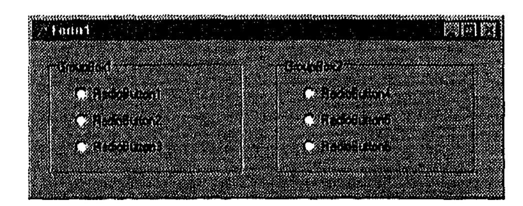
لإنشاء هذه النافذة :

١- أضف أثنان من الـ GroupBox الاثنين اولاً .

٢- اضف RadioButtons الى الـ GroupBox objects .

٣- ان ضـغط مـؤشـر الفـأرة داخل الـ GroupBox لإدخـال الـ RadioButtons (أو أى control آخر) يخبر Delphi ان يربط الـ object الذى تم إدخاله بالـ GroupBox .

حتى يعمل عـملية الـ tab بشكل صـحيح ، يجب اختيار RadioButton واحـد على الاقل من كل مـجـمـوعـة . حـدد خـاصـيـة الـ Checked بـ True لـ RadioButton واحد فى كل GroupBox . وهذا من حدود الـ Windows على radio button controls المجموعة .



شكل (٦-٧): تستخدم الـ GroupBoxes لترتيب RadioButtons فى مجموعة منطقى

اذا أضفت RadioButtons قبل ان تنشئ الـ GroupBoxes الخاصة بهم، لن تعمل الازرار كمجموعات متميزة. ان سحب الـ RadioButtons فى الـ GroupBoxes لن يكون مفيداً - تبقى objects مرتبطة بالـ components التى تم إسقاطها عليها فى أول الأمر (الـ form أو control يحتوية آخر). اذا واجهت هذه المشكلة، اختر الـ RadioButtons، وأحفظهم فى الـ clipboard اختر الـ intended GroupBox، والصق الـ RadioButtons من الـ clipboard إلى الـ container الجديدة يمكنك ايضاً حذف الازرار وإعادة إدخالها من على VCL palette مباشرة الى GroupBox، ولكن هنا تفقد اية تغييرات فى الخاصية تكون قد صنعتها للـ controls.

لجمع radiobuttons أو control objects آخر فى وقت التشغيل، حدد خاصية الـ Parent بالـ GroupBox للزر المختار. (يمكنك ان تفعل هذا فى وقت التشغيل فقط لان خاصية الـ Parent لا تكون متاحة فى نافذة الـ Object Inspector). على سبيل المثال، لربط ثلاثة RadioButton objects بالـ GroupBox1، استخدم عبارة مثل التالية، ربما فى OnCreate للـ form:

```
RadioButton1.Parent := GroupBox1;
RadioButton2.Parent := GroupBox1;
RadioButton3.Parent := GroupBox1;
RadioButton1.Checked := True;  { Select one grouped button }
```

والطريقة الصائبة لتعرف اذا كانت الـ RadioButtons قد تم تجميعها بشكل سليم هى ان تسحب الـ GroupBox. اذا كانت الـ RadioButtons مثل البط الصغير الذى يتبع أمه، فيكون قد أتم بهذه المجموعة. اذا لم يكن كذلك، ففى الغالب يكون الازرار قد تم ربطها بالـ form. قص وإلصق الازرار لحل المشكلة.

يمكنك ايضاً تجميع الـ GroupBox ، ولكن لان هذه الـ controls تعمل منفردة، فلا يوجد غالباً سبب وجيه لكى تجمعها فى مجموعة. فى غالبية الحالات، يكن من الجيد استخدام الـ Bevelsأو الـ Panels كما هو الحال فى ترتيب مجموعات بصرية من الـGroupBox.

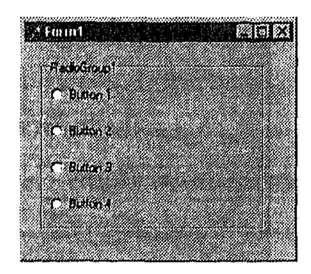
لكى تجعل مجموعات RadioButtons تعمل كما تريدها عندما يضغط المستخدمون مفتاح الـ Tab، فعليك ان تعبث فى نظام الـ Tab الخاص بالـ form بعد إنشاء الـGroupBox وبإضافة الـ RadioButton، اختر الـ form واختر Edit|Tab Order لتحديد الترتيب الذى بموجبه يستطيع المستخدمون ضغط الـ Tab للتنقل بين المجموعات. بعد ذلك، اختر كل GroupBox، ومرة أخرى استخدام الـ Edit|Tab Order، ولكن هذه المرة، للتأثير على ترتيب Tab الـRadioButtons المجمعة.

استخدام الـ RadioGroups:

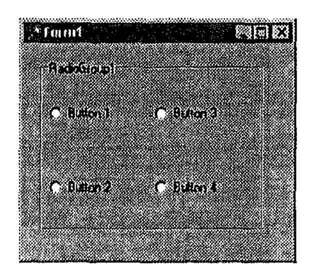
يدرك غالبية المطورين ان الـ GroupBoxes والـ RadioButtons لها طبيعية مشاكسة كالثيران فى الحلقة. والطريقة الاقل جهداً لإنشاء مجموعات RadioButtons، أضف RadioGroup component على الـform. لا تدخل الـ RadioButtons فى الـ RadioGroups. بدلاً من ذلك، اختر خاصية الـ Items للـ RadioGroups، واضغط قيمتها مرتين أو اضغط الزر البيضاوى لفتح الـ String list editor فى الـ Delphi.

اضف labels لكل radio button بالترتيب الذى تريده ان تظهر فيه. هذا هو كل ما عليك ان تفعله لإنشاء مجموعات من الـ RadioButton فى الـ RadioGroup.

واختيارياً حدد خاصية الـColumns للـ RadioGroup بعدد الاعمدة لعرض الـRadioButtons. يوضح الشكل (٦-٨) اربعة RadioButton فى RadioGroup مع الـ Columns مساوية الى (1). يوضح شكل (٦-٩) نفس الـ RadioGroup مع الـ Columns محددة بـ 2.



شكل (٦-٨): اربعة RadioButton
فى RadioGroup والـ Columns محددة (1)



شـــكل (٦-٩): نفـس الازرار مـن
شكل (٦-٨) مع الـ Columns مساوية لـ (2)

استخدم خاصية الـ ItemIndex لفحص الزر الذى تم اختياره حالياً. حدد الـ ItemIndex بـ (-1) "وهى القيمة الأفتراضية) لإبطل اختيار جميع الازرار فى مجموعة ما. حددة بـ (0) لاختيار الزر الأول، وبـ 1 لاختيار الزر الثانى، وما الى ذلك.

يمكنك إنشاء radio buttons فى RadioGroup فى وقت التشغيل. على سبيل المثال، استخدم هذا المتغير والـ procedure للسماح للمستخدمين بإضافة radio buttons جديدة فى مجموعة :

```
var
    S: String;
begin
    if InputQuery('Input', 'Enter Radio Button', S) then
      RadioGroup1.Items.Add(S);
end;
```

الـ Items array عبارة عن string list من نوع الـ TStrings . إستدع Add method الخـاص بالقـائمـة لإدخـال string جـديد ، والذى يجـعل ايضـاً الـRadioGroup ينشئ RadioButton جديد مستخدما الـstrings كـ label .
وتسـاوى خـاصيـة الـ Items Count عـدد strings فى خـاصيـة الـ strings ، ولذلك ، فـهى ايضـاً تسـاوى عـدد الـ RadioButtons الذى تملكه الـ RadioGroup . وتسـاوى الـ ItemIndex ، الـ button's index الذى اختيـاره حالياً .

استخدم هذه القيم للتكرار من خلال button labels ، كما فى العبارة التالية ، التى تستدعى AnyProcedure (غير موضح) :

```
for I := 0 to RadioGroup1.Items.Count _ 1 do
  AnyProcedure(RadioGroup1.Items.Strings[I]);
```

أو ، استخدم عبارة with لجعل الـcode السابقة اكثر وضوحاً ، ولتقليل الإشارة الى RadioGroup1 من اشارتين الى واحدة :

```
with RadioGroup1, Items do
  for I := 0 to Count - 1 do
      AnyProcedure(Strings[I]);
```

لاحظ الـ syntax الخاصة لعبارة الـ with ، باستخدام الفصلات لفصل أكثر من identifiers ان السطر الأول فى المثال السابق مساو لـ :

```
with RadioGroup1 do with Items do
```

استخدم الـ ItemIndex لتحديد الزر الذى تم اختياره . فمثلاً ، عبارة if يمكن ان تقوم بعمل لزر معين معرف بواسطة الـ label الخاص به :

```
with RadioGroup1, Items do
  if Strings[ItemIndex] = 'Button2' then...
```

حدد دائماً زراً واحداً بحالة فتحه لكل RadioGroup ؛ والا ، لا يمكن للمستخدمين ان يضغطوا الـ Tab لتحويل الـ focus الى المجموعة (إحدى غرائب الـ Windows) . إمـا ان تحـدد الـItemIndex بـ (0) عندمـا تصمم الـ RadioGroup ، أو تدخل الـ code التالية فى OnCreate الخاص بالـ form :

```
with RadioGroup1, Items do
if Count > 0 then
  ItemIndex := 0
else
  ItemIndex := -1;
```

يمكنك ايضاً استخدام هذا الـ code للـ RadioGroup الذى ينشئه البرنامج فى وقت التشغيل؛ فى مثل هذه الحالة يجب ان تسمح بامكانية ان تكون المجموعة بلا ازرار. فى هذه الحالة، يجب ان تحدد الـ ItemIndex بـ 1- كما هو موضح هنا.

تحتوى الـRadioGroup على قائمتين تمثلا الـ RadioButtons المجمعة. يحتوى Items arrays على string list للـ button labels وهى متاحة فى وقت التصميم ووقت التشغيل و Components array، المتوفر فى وقت التشغيل فقط، يحتوى على إرشادات عن الـ RadioGroup objects الحقيقية.

افحص الـ Components array والعناصر المتعلقة به، اضف RadioGroup object على الـ form، اضف. button labels متعددة فى Item's string list. أضف Button وقم بإنشاء OnClick مع هذه العبارات:

```
with RadioGroup1 do
if ItemIndex >= 0 then
  ShowMessage('You selected ' + Items.Strings[ItemIndex]);
```

قم بتشغيل البرنامج واضغط الزر لعرض radio button's label التى تم اختيارها حالياً. اترك البرنامج وارجع الى Delphi، ثم حدد Breakpoint على عبارة الـ with. لفعل هذا، اضغط مرة واحدة مؤشر الفأرة فى اقصى يسار السطر، أو حرك مؤشر لوحة المفاتيح الوامض الى السطر واختر Run|Add Breakpoint. فى أى من الطريقتين، يلون Delphi العلامة العلوية للعبارة باللون الاحمر، والتى تشير الى ان البرنامج سوف يتوقف قبل ان تتنفذ هذه العبارة مباشرة.

قم بتشغيل البرنامج مرة اخرى واضغط الزر. يتوقف البرنامج عند breakpoint. حرك المؤشر الوامض فى أى مكان داخل الـ RadioButton1، واضغط Ctrl+F7 لفتح Evaluate/Modify dialog الخاص بـ debugger. يجب ان ترى الـ RadioGroup1 فى مجال الـ Expression وقائمة يقيم

الخـصـائص فى مـربع الـResult. اذا لم يكـن كـذلك، إدخل RadioGroup1 فى حقل الـ Expression واضغط Evaluate. وتعتبر هـذه تقنية مفيدة تضع قائمة باعـضـاء الـ objects الغـيـر مـتـوفـره بشكل عـام من خـلال نافـذة الـ Object Inspector.

وتعتبر القوائم المطولة عن القيم فى الـ objects والـ records صعبة فى فك شفرتها فى الـ Evaluate/Modify dialog. لعرض كل خاصية بإسمها وقيمتها، إدخل ",R" (لا تدخل علامات التنصيص، ولكن لا تهمل الفصلة قبل الـ R) بعد الـ identifier فى حقل الـ Expression. على سبيل المثال، اذا كنت متبعاً، اكتب تعبير RadioGroup1,R واضغط Enter.

Tip

فكرة: قم بتشغيل tool-tip expression evaluation باستخدام الـ Tools|Environment Options و Code Insights page tab. يمكنك عندئذ نقل المؤشر الى اسم object مثل RadioGroup1، توقف قليلاً، وانظر ان قيمة الـobject معروضة فى pop-up window صغيرة.

يجب ان ترى الآن القيم التـاليـة، بالإضـافـة الى اخـرى كـثـيـرة، فى الـ Evaluate/Modify dialog. اضـغط Evaluate اذا لم ترى ذلك. يتـبـع اسم الخاصية الـ class method المتعلقة بها. فمثلاً، ها هنا أول سطرين من حقل الـ Result:

```
Components:<GetComponent>;
ComponentCount:<GetComponentCount>;
```

يحـتـوى الـ Components على RadioButton objects تنتـمى الى الـRadioGroup. تسـاوى الـ ComponentCount عـدد هذه الـobjects. استـخـدم هاتين القـيـمـتين للتـوصـيل الى RadioButtons منفـردة فى الـ RadioGroup. على سبيل المثال، لتحديد خاصية الـ Checked بـ False لكل الـ RadioButtons فى مجموعة، يمكنك استخدام for loop مثل هذه:

```
with RadioGroup1 do
for I := 0 to ComponentCount - 1 do
  TRadioButton(Components[I]).Checked := False;
```

أو، لاختيار معينا، استخدم هذه العبارة:

```
with RadioGroup1 do
  TRadioButton(Components[2]).Checked := True;
```

عند تذكر الـComponents array، يجب ان تخبر الـcompiler أى نوع من الـ objects يحتوى عليه الـ array. افعل هذا باستخدام اسم الـ class (هنا TRadioButton) وكأنه function تصف object من نوع محدد. قم بتمرير indexed Components reference لإقامة هذا النوع. يمكنك عندئذ التوصل الى خصائص object مثل Checked.

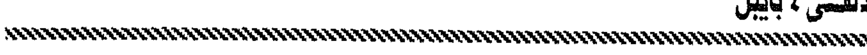
اذا كنت تتابع، إرجع الى Delphi، واضغط F9 لاستمرار عمل البرنامج، الذى توقف عند الـ breakpoint أوقف عمل البرنامج للعودة الى نمط التصميم فى Delphi. يمكنك عندئذ إزالة الـ breakpoint، أو بدء مشروع جديد.

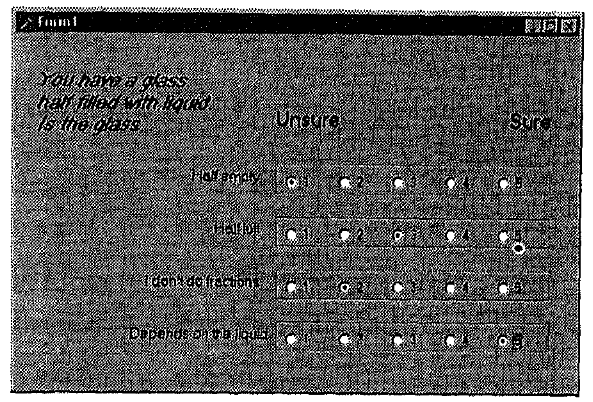
وتفيد الـ RadioGroups ايضاً فى تصميم form استطلاع الرأى، option form، multiple choice form. قم بإنشاء RadioGroup متعددة، وادخل مسافات خالية، علامات ترقيم، خانات رقمية، أو رموز اخرى فى Items array فى هذه الحالة، تكون فى الغالب لا تريد labels كاملة- إنك تريد ان تعرض فقط دوائر الـ RadioButton control. حدد خاصية الـ Columns مساوية لعدد الازرار، قلل عدد الـ RadioGroup قدر ما تستطيع. يوضح شكل (٦-١٠) مثال افتراضى يستخدم الـ RadioGroup لجانب سيكولوجى استوحيته من صديق طبيب نفسى الذى يقوم باختبارات مشابهة.

بعض النسخ من وثائق Delphi تقول انه بامكانك وضع انواع اخرى من الـ controls (بخلاف radio buttons) فى الـ TRadioGroup هذا غير صحيح. يمكنك إضافة RadioGroup فقط فى الـ RadioGroup.

انواع اخرى من الـ RadioGroup:

قد تعرف انه بإمكانك إضافة الـ RadioButton مباشرة فى نافذة الـ form. اذا كنت تريد مجموعة واحدة فقط من radio buttons، فلا يجب عليك إدخالها فى GroupBox ولا تحتاج أن تستخدم RadioGroup. فقط أدخل ما تستطيع إدخاله من RadioButton على قدر حاجتك فى أى form.





شكل (٦-١٠): تساعد الـ RadioGroup فى إنشاء اختبارات متعددة الاختيارات واستطلاعات رأى كما فى هذا الجانب الإفتراضى

ومن غير المعروف، (ولكنه ليس سراً)، أنه بإمكانك أيضاً تجميع الـ RadioButton المتعددة فى Panel. قم بتجربة هذا:

١- أضف Panels على الـ form، ثم أضف الـ RadioButtons المتعددة فى كل panel.

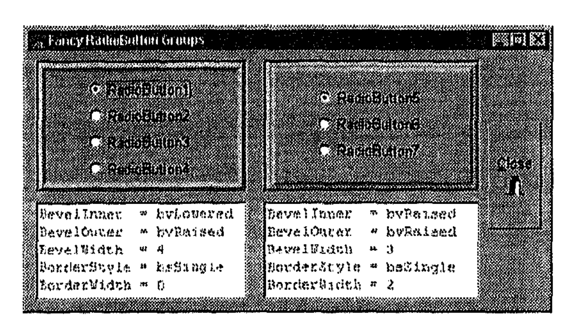
٢- حدد خاصية الـ Checked لأحد الـ RadioButton فى كل مجموعة بـ True.

٣- قم بتشغيل البرنامج.

كما توضح هذه التجربة، لا يوجد إختلافات عملية بين الـ GroupBox والـ Panel- على الأقل من ناحية قدرتها على تجميع الـ radiobuttons المتعددة. ولكن يظهر تعليق الـ GroupBox على حده العلوى- يعرض الـ Panel تعليقه فى الداخل، وقد تحتاج أن تحذفه لتفسح مكاناً لـ radiobuttons والـ controls الأخرى.

على القرص المدمج: إحدى الميزات فى استخدام Panels كمربعات مجموعة هى إمكانية الجرافيك 3D الخاص بالـ Panel، والتى يفتقدها الـ GroupBox. كما يوضح شكل (٦-١١)، يمكنك استخدام الـ Panels لإنشاء مجموعات RadioButton خيالية. والنص الموجود تحت كل مجموعة من الـ RadioButtons يوضح قيم الخاصية التى تنتج تأثيرات الـ 3D الموضحة.

وتطبيق الـ Fancy الذى ليس له code هام، موجود على دليل الـ Source\Fancy فى القرص المدمج المرفق بهذا الكتاب .



شكل (٦-١١): التطبيق Fancy يظهر كيفية استخدام الـ Panel لإنشاء مجموعات 3D RadioButton

من الممكن إضافة objects من انواع مختلفة فى الـGroupBox والـ Panel . على سبيل المثال، يمكن ان يحتوى الـ Panel على ثلاثة من الـ RadioButtons واثنين من الـ CheckBoxes . يمكن للمستخدمين ان يضغطوا Tab للتنقل بين المجموعات . والنتيجة النهائية قد يكون مفيدة فى البرامج التى تحركها لوحة المفاتيح الى حد كبير . ولكن لان المستخدمين يختارون الـ CheckBoxes منفردة، فإن إضافتها للـ Panel أو الـ CheckBoxes ليس له ميزة حقيقية .

Spin Buttons :

تجد الـ Samples page tab على VCL palette، button controls إضافيتين، وهما الـ SpinButton والـ SpinEdit . قم بتعيين الـ glyphs الخاصة بك لخصائص الـ DownGlyph فى الـ UpGlyph التابعة لهذه الـ objects لعرض اسهم مخصصة مشيرة الى أعلى أو أسفل أو لعرض objects أخرى . والـ components هما :

- **SpinButton:** مزدوج ذو أزرار علوية وسفلية . إن ضغط أحد الأزرار يستدعى OnUpClick أو OnDownClick .

- **SpinEdit:** هو SpinButton ملحق بـ edit field إن ضغط السهم العلوى أو السفلى للزر يزيد أو ينقص قيمة صحيحة فى edit field . يمكن للمستخدمين أن

يدخلوا قيـم فى هذا الحـقل، والذى يتعـرف ايضـاً على اوامـر الـ cut، والنسخ، واللصق.

استخدام SpinButton components :

لإستخدام SpinButton components إتبع هذه الخطوات :

١- أضـف الـ SpinButton عـلـى الـ form واضـف الـ code فـى OnUpClick و OnDownClick الخـاصـة به. اضف الـ Label على الـ form وقم بتعريف المتغير التالى فى قطاع الـ TForm1 class's private :

```
private
  Count: Integer;
```

٢- قم بإنشـاء الـ OnCreate الخـاص بالـ form وابدأ الـ Count بصـفـر : .zero

```
Count := 0;
```

٣- قـم بإنشـاء الـ OnUpClick و الـ OnDownClick الخـاصـة بالـ SpinButton's . استخدم البرمجة الموجودة فى القائمة (٦-١) لزيادة أو نقص الـ Count، ولعرض قيمة الـ Count فى الـ label .

القائمة (٦-١): استخدم الـ SpinButton وهذه البرمجة فى الـ OnUpClick و الـ OnDownClick لزيادة وخفض متغير الـ Count وعرضه فى الـ label

```
procedure TForm1.SpinButton1DownClick(Sender: TObject);
begin
 Dec(Count);
   Label1.Caption := IntToStr(Count);
end;

procedure TForm1.SpinButton1UpClick(Sender: TObject);
begin
 Inc(Count);
   Label1.Caption := IntToStr(Count);
end;
```

استخدم خصائص الـ UpGlyph والـ DownGlyph في الـ SpinButton لعرض glyphs للازرار العلوية والسفلية الخاصة بالـ object إما ان تختار من مخزن Delphi، bitmaps للاسهم، أو تنشئ glyphs خاصة بك.

قد لا تحتوى glyphs لـ SpinButton على صور متعددة، ولذلك، فإن استخدام bitmaps المقدمة من Delphi (والتى تحتوى جميعها على صور مزدوجة)، يجب عليك ان تحولها الى glyphs احادية الصورة افعل هذا بنسخ الملفات التى تريدها على دليل آخر - ملفات الـ Arrow1d.bmp والـ Arrow1u.bmp مثلاً من دليل Images\Buttons الخاص بـ Delphi. افتح هذه الملفات فى الـ Image Editor من قائمة الـ Tools. لكل ملف، اختر Image|Attributes، قم بتغيير الـ Width من ٣٢ الى ١٦ (أو إجعله مساوياً للـ Height)، وأبطل إختيار صورة الـ Scale حتى تتلائم. يمكنك عندئذ حفظ الصورة التى تم تحويلها. إن هذه هى أسهل طريقة وجدتها لتقليص الـ glyph متعدد الصور ليصبح bitmap لصورة واحدة.

بالقطع يمكنك أيضاً إنشاء الـ glyphs الخاصة بك باستخدام أى bitmap editor فى أى من الحالتين، يمكنك الآن إدخال الـ glyphs الخاصة بك فى SpinButton. على سبيل المثال، جرب هذه الخطوات:

١- أضف الـ SpinButton من الـ Samples page tab على الـ VCL palette. تأكد من إختيار الـ SpinButton وليس الـ SpinEdit.

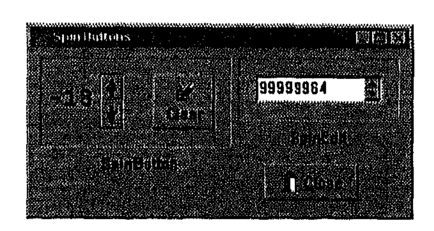
٢- اضغط الزر البيضاوى لخاصية الـ DownGlyph لفتح الـ Image Editor. قم بتحميل ملف صورة سهم سفلى معدل مثل الـ Arrow1d.bmp.

٣- أعد الخطوة الثانية مع خاصية الـ UpGlyph لتحميل Arrow1u.Bmp معدل.

٤- اعتماداً على حجم glyph bitmap، قد يكون عليك أن تعدل أبعاد الـ SpinButton لإظهار الـ glyph كاملة.

استخدام الـ SpinEdit components:

على القرص المدمج: إن الـ SpinEdit يجمع بين SpinButton و Edit input field لإنشاء زر إسطوانى ذا إمكانيات editing كاملة، بما فى ذلك أوامر القطع والنسخ واللصق. يوضح شكل (٦-١٢) الـ SpinEdit و SpinButton من تطبيق الـ SpinButt على القرص المدمج فى دليل الـ Source\SpinButt . توضح القائمة (٦-٢) الـ source code للتطبيق.



شكل (٦-١٢): التطبيق SpinButt يوضح SpinEdit object والـ SpinButton

ملحوظة: لقد استبدلت الـ SpinEdit object والـ SpinButton بـ UpDown الجديد نسبياً. قد تريد استخدام الـ UpDown بدلاً من الـ SpinButton والـ SpinEdit، إلا اذا كنت تستخدم نسخ أولية من Delphi، أو اذا كنت تحتاج أن تقدم تجانس خلفى. ولكن، لا يجد شئ خطأ فى هذه الـ components ولا يجب أن تتردد فى استخدامها اذا وفرت لك الخصائص التى تريدها. بعد الفصل القادم، سوف اشرح كيفية استخدام الـ UpDown.

القائمة (٦-٢): Spinbutt\Main.pas

```
unit Main;

interface

uses
  SysUtils, Windows, Messages, Classes, Graphics, Controls,
    Forms, Dialogs, Spin, StdCtrls, ExtCtrls, Buttons;

type
    TMainForm = class(TForm)
```

```
    SpinButton1: TSpinButton;
      Label1: TLabel;
      SpinEdit1: TSpinEdit;
      SpinEdit: TLabel;
      SpinLabel: TLabel;
      BitBtn1: TBitBtn;
      Bevel1: TBevel;
      Bevel2: TBevel;
      BitBtn2: TBitBtn;
      procedure FormCreate(Sender: TObject);
      procedure SpinButton1DownClick(Sender: TObject);
      procedure SpinButton1UpClick(Sender: TObject);
      procedure BitBtn1Click(Sender: TObject);
      procedure SetSpinButtonCaption;
    private
     { Private declarations }
     Count: Integer;
    public
   { Public declarations }
    end;

var
    MainForm: TMainForm;

   implementation

   {$R *.DFM}

 const
    minCount = -99;
     maxCount = 99;

 procedure TMainForm.SetSpinButtonCaption;
 begin
  SpinLabel.Caption := IntToStr(Count);
```

```
end;

procedure TMainForm.FormCreate(Sender: TObject);
begin
 Count := 0;
end;

procedure TMainForm.SpinButton1DownClick(Sender: TObject);
begin
 if Count > minCount then Dec(Count);
    SetSpinButtonCaption;
end;

procedure TMainForm.SpinButton1UpClick(Sender: TObject);
begin
 if Count < maxCount then Inc(Count);
    SetSpinButtonCaption;
end;

procedure TMainForm.BitBtn1Click(Sender: TObject);
begin
 Count := 0;
    SetSpinButtonCaption;
end;

end.
```

إن الـSpinEdit له شكل Button داخلي من نوع الـ SpinButton . (إن الاسم Button يعتبر إختيار سيء- إنه ليس Button component ، ولكن object من الـ SpinButton يدعى Button). استخدم الـ Button الخاص بالـ SpinEdit كما تفعل في حالة الـ SpinButton المنفرد. على سبيل المثال، لتعيين صور glyph علوية وسفلية من نوع الـ TBitmap ، يمكنك إدخال عبارات مثل هذه:

```
SpinEdit1.Button.UpGlyph := NewUpGlyph;
SpinEdit2.Button.DownGlyph := NewDownGlyph;
```

الـStatic Text أو (TStaticText):

إن Delphi يقدم component جديد نسبياً لعرض الـ labels، وهو الـ StaticText، على Additional palette . ومثل الـ Standard Label، يعتبر الـ StaticText مفيداً في إنشاء الـinteractive labels يمكن للمستخدم أن يضغطها بالفأرة.

ملحوظة: من الناحية الفنية، إن TStaticText class تشبه الـ TLabel، فيما عدا أن الـ TStaticText تنحدر من الـ TWinControl. بمعنى آخر، إن الـ Label ليس لديها window handles؛ ولكن الـ StaticText لديها. في حالة ضرورة أن ينتمي الـ accelerator key الى control لديه window handle، فعليك استخدام الـ StaticText بدلاً من الـ Label.

إن الـ StaticText له ثلاث خصائص غير موجودة في الـ Label. والخصائص الجديدة هي:

- **BorderStyle:** حدد هذه بـ sbsNone، أو بـ sbsSingle، أو بـ sbsSunken . بإستثناء الـ sbsNone، قد تريد تحديد الـ AutoSize بـ False؛ وإلا؛ يتم تحديد حجم الـ control تلقائياً إعتماداً على كم نص الـ Caption الذى يعرضه.
- **TabOrder:** حدد هذه الخاصية حسب الـ tab order لبند الـ StaticText. ولان الـcomponent له window handle ، فإنه قد يتلقى input focus (بالرغم من ان المستخدمين لا يستطيعون الكتابة داخله).
- **TabStop :** حدد هذه بـ True لإضافة StaticText الى قائمة الآخرين الذين يتلقون الـ input focus، عندما يضغط المستخدمون الـ Tab.

إثنين من انماط الـ Label غير موجودين في الـ StaticText وهما:

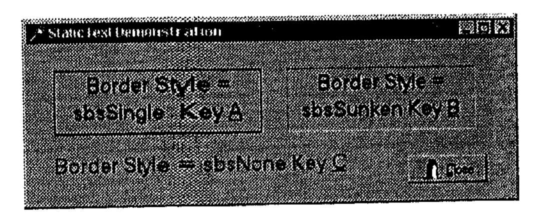
- **Transparent :** عندما تحدد هذه الخاصةبـ True، فإن خلفية الـ Label لا تغير أى control أو نافذة أخرى. وبالارتباط بخاصية الـ Color، يمكنك تحديد

الـ Transparent بـ False لتلون الخلفية بوضوح (والذى يعرض النص اسرع عن كون الـ Transparent محدد بـ True) . ان الـ StaticText لا تكون ابداً Transparent ، ولذلك فهى تفتقد هذه الخاصية . ولكن ، لإنشاء StaticText يبدو شفافاً ، ببساطة حدد خاصية الـ Color له بنفس الطريقة وكان object ما موجود تحت النص .

• **WordWrap :** مع الـ Label ، يمكنك اختيارياً ان تحدد هذه الخاصية بـ True لقطع الخطوط الطويلة من النص بين الكلمات تلقائياً . ودائماً ما تقوم الـ StaticText بضبط النص الطويل داخل حجمه المحدد . حدد الـ AutoSize بـ False عند عرض نص طويل فى الـ StaticText ؛ والا ، لن يضبط الـ control النص الخاص به ولكن بدلاً من ذلك سوف يعرض خطاً واحداً ، بغض النظر عن طوله .

On the CD-ROM

على القرص المدمج: ان الـ StaticText يفيد فى إنشاء text objects يمكن للمستخدمين اختيارها بالضغط بالفأرة . يوضح تطبيق الـ StaticDemo فى القائمة (٦-٣) كيفية فعل هذا ، ويظهر ايضاً الانماط الثلاثة المختلفة لـ StaticText . يوجد البرنامج على القرص المدمج فى دليل الـ Source\StaticDemo . يوضح الشكل (٦-١٣) الناتج عن تنفيذ البرنامج .



شكل (٦-١٣): يظهر تطبيق الـ StaticDemo انماط الـStaticText، كما يظهر كيفية إنشاء عناصر نص يمكن للمستخدمين ضغطها بالفأرة

القائمة (٦-٣): taticDemo\Main.pas

```
unit Main;

interface
```

```
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
    Forms, Dialogs,  StdCtrls, Buttons;

type
  TMainForm = class(TForm)
      StaticText2: TStaticText;
        StaticText1: TStaticText;
        StaticText3: TStaticText;
        BitBtn1: TBitBtn;
        procedure StaticText1MouseDown(Sender: TObject;
          Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
            procedure StaticText1MouseUp(Sender: TObject;
          Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
        procedure StaticText1Click(Sender: TObject);
    private
      { Private declarations }
       public
      { Public declarations }
       end;

var
    MainForm: TMainForm;

implementation

{$R *.DFM}
var
    SavedStyle: TStaticBorderStyle;

{ Change static text border style to indicate its selection }
```



```
procedure TMainForm.StaticText1MouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  with Sender as TStaticText do
     begin
      SavedStyle := BorderStyle;
         if BorderStyle = sbsSunken
          then BorderStyle := sbsSingle
             else BorderStyle := sbsSunken;
          end;
end;

{ Reset static text border style when mouse button released }
procedure TMainForm.StaticText1MouseUp(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  with Sender as TStaticText do
        BorderStyle := SavedStyle;
end;

{ Make a sound when text object is clicked }
procedure TMainForm.StaticText1Click(Sender: TObject);
var
    S: String;
begin
    MessageBeep(0);
    S := TStaticText(Sender).Name;
    ShowMessage('You selected ' + S);
end;

end.
```

يوضح برنامج العرض إحدى الطرق لإعطاء إرجاع بصرى عندما يضغط المستخدمون. StaticText . لقد تم تعيين MouseDown و MouseUp لـ StaticText الثلاثة فى الـ form، كلاً منها له قيمة خاصية BorderStyleمختلفة .

عندما يضغط المستخدم الفأرة على أحد الـ StaticText، يحفظ MouseDown handler الـ BorderStyle الحالى فى global variable ثم يغير النمط الى شئ آخر. عندما يتلقى المستخدم تأكيداً بصرياً بأن الـ control يتعرف على الضغط على الفأرة. عندما يطلق المستخدم الفأرة، يعيد MouseUp تحديد خاصية الـ BorderStyle بقيمتها المحفوظة، والتى تستعيد المظهر الاصلى للـ control.

فى كلتا الحالتين، من الضرورى ان تستخدم عبارة مثل التالية :

```
with Sender as TStaticText do
  BorderStyle := SavedStyle;
```

ان Sender parameter الذى تم تمريره للـ event handler يساوى الـ object الذى ضغطه المستخدم. ولان الـ form ايضاً لها خاصية BorderStyle، والتى هى من نوع آخر، من الضرورى ان تستخدم عبارة with أو أى مشير آخر لتخبر الـ compiler أى من خاصية الـ BorderStyle تستخدم. والطريقة لفعل هذا هو ان تسخدم عبارة مثل :

```
StaticText1.BorderStyle := sbsSunken;
```

بدون اسم الـ object والنقطة، يعتبر الـ BorderStyle غامضاً عند استخدام خاصية الـ BorderStyle فى عبارة برنامج، اذا تلقيت رسالة الخطأ "Incompatible types: 'TFormBorderStyle' and 'TStaticBorderStyle'," ، استخدم احدى التقنيتين السابقتين لحل التداخل .

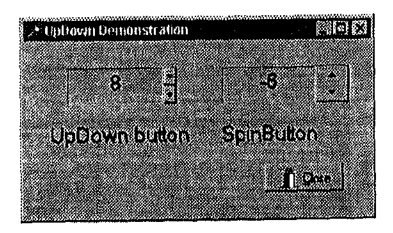
والـ event handler الثالث والأخير فى التطبيق يعطى نغمة للإشارة الى ان الـ StaticText قم تم اختياره، ويعرض ايضاً رسالة تحتوى على اسم الـ object. وهذا يظهر كيف يمكنك استخدام خاصية الـ Sender لتحديد أى بند من عناصر الـ StaticText قام المستخدم باختياره. أو، يمكنك برمجةOnClick منفصلة لكل StaticText.

: Up-Down Button Controls

وهناك component آخر جديد نسبياً يجعل من السهل إنشاء ازرار ذات اسهم علوية وسفلية ، والتى تستخدم لاختيار قيم أو تحريك مجموعة من العناصر فى edit window أو label وكانت الطريقة الوحيدة لفعل هذا فى النسخ الأولى من Delphi هى باستخدام الـ Sample SpinButton ، الموضح سابقاً فى هذا الباب . ولكن ، لان هذا مجرد Sample ، وليس من المعايير الرسمية لـ Delphi ، كان المبرمجين غير مستعدين لاستخدامه خوفاً من ان الـ Borland قد لا يدعم الـSpinButton فى المستقبل .

وإننى اؤكـد ان الـ UpDown على الـ Win32 يفـوق الـ SpinButton ، وإننى ارشح استخدام الـ UpDown من الآن فصاعدا . فإنه يسهل ربطهم بالـ controls الاخـرى مـثل الـ StaticText والـ Edit ، وكـ Windows controls اصلية ، فإنهم يضفون مظهراً متماشياً مع واجهات تطبيق البرمجيات الاخرى .

On the CD-ROM **على القرص المدمج:** لمقارنة الـ UpDown و الـ SpinButton كتبت تطبيق الـ UpDown على القرص المدمج . يوضح شكل (٦-١٤) عرض التطبيق . توضح القائمة (٦-٤) الـsource code . افتح ملف مشروع الـ UpDownDemo.dpr واضغط F9 لتشغيل البرنامج فى Delphi . اضغط الازرار الى اليسار من عناصر الـ StaticText يرتبط الـ UpDown تلقائياً بالـ text object الخاص به ؛ يتطلب الـ SpinButton برمجة ليفعل نفس الشئ .



شكل (٦-١٤): يقارن تطبيق الـ UpDownDemo الـ UpDown الاصلى فى الـ Win32 بـ SpinButton الخاص بـ Delphi على Samples palette.

القائمة (٦-٤): UpDownDemo\Main.pas

```
unit Main;
```

```
interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
    Forms, Dialogs, StdCtrls, Spin, ComCtrls, Buttons;

type
  TMainForm = class(TForm)
      UpDown1: TUpDown;
        SpinButton1: TSpinButton;
        Label1: TLabel;
        Label2: TLabel;
        BitBtn1: TBitBtn;
        StaticText1: TStaticText;
        StaticText2: TStaticText;
        procedure SpinButton1UpClick(Sender: TObject);
        procedure SpinButton1DownClick(Sender: TObject);
      private
    { Private declarations }
      public
    { Public declarations }
      end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}
```

```
{ These constants and the two event handlers are needed
  by SpinButton components to associate them with another
  control, a StaticText object in this demonstration. The
  new UpDown component needs none of this programming
because
  it can be automatically associated with another control. }
const
  Min = -10;
    Max = +10;

{ Respond to user clicking the SpinButton's Up button }
procedure TMainForm.SpinButton1UpClick(Sender: TObject);
var
  V: Integer;
begin
  V := StrToInt(StaticText2.Caption);
    if V = Max
      then V := Min
        else inc(V);
        StaticText2.Caption := IntToStr(V);
end;

{ Respond to user clicking the SpinButton's Down button }
procedure TMainForm.SpinButton1DownClick(Sender: TObject);
var
  V: Integer;
begin
    V := StrToInt(StaticText2.Caption);
    if V = Min
      then V := Max
        else dec(V);
        StaticText2.Caption := IntToStr(V);
end;

end.
```

لاشئ من البرمجة الموجودة فى القائمة (٦-٤) يحتاج إليه مع UpDown controls- فهم يسهل ربطهم مع الـ text labels والـ edit field. على سبيل المثال، فى هذه الحالة، لربط الـ UpDown مع عنصر النص الخاص به على اليسار، فقد حددت خاصية الـ Associate لـ UpDown1 بـ StaticText1. ويتولى الـ control أمر جميع تفاصيل العرض الخاصة بتحويل القيم العددية الى strings وعرضها فى نافذة النص. هذه الخطوات تتم يدوياً لـ SpinButton كما هو موضح فى القائمة:

خصائص الـ UpDown:

ويقدم الـ UpDown ايضاً عدداً من الخصائص المتعة التى يمكنك تجربتها واهم هذه الخصائص هى:

- **AlignButton**: حدد هذه الخاصية بـ udLeft أو udRight للصق الازرار باحد جوانب أو الجانب الآخر للـ control المرتبط بها. وهذه الخاصية ليس لها تأثير الا اذا كانت الـ Associate محددة باسم الـ control الآخر. عند استخدام هذه الخاصية، لن تنجح محاولات تغيير موضع ازرار الـ UpDown يدوياً بالنسبة للـ control المرتبط بها.

- **ArrowKeys**: عند ما تحدد بـ True، تقوم هذه الخاصية بتشغيل مفاتيح الاسهم المشيرة لاعلى ولاسفل فى لوحة المفاتيح لضغط ازرار الـ control يجب للـ UpDown ان يكون input focus (حدد الـ TabStop بـ True).

- **Associate**: حدد هذه الخاصية باسم الـ object الآخر، والذى سيكون فى الغالب Label، أو StaticText، أو Edit. استخدم الـ AlignButton لوضع الـ UpDown على احد جانبى الـ control المرتبط به. ان ربط control بـ UpDown هو كل ما تحتاج فعله لتوفير تتابع آلى للقيم فى الـ control .

- **Cursor**: اختر شكل مؤشر اذا كنت تريده ان يتغير عندما يوجه المستخدمين الفأرة الى ازرار للـ UpDown.

- **Increment**: فى كل مرة بضغط المستخدم زراً تتغير قيمة الـ UpDownعلى حسب القيمة المحددة فى الـ Increment.

● **Max, Min** : ضع حـدأ لمدى الـ control بتـحـديد هـاتين القـيـمـتين . في العـادي ، يجب ان تكون الـ Min أقل من الـ Max ؛ ولكن ، اذا كـانت الـ Min أكبر من الـ Max ، يبدو الـ UpDown وهو يعمل الى الوراء (اضغط الزر العلوى مثلاً يخفض قيمة الـ control) اذا لم تكون تريد إرباك مستخدمي برنامجك ، تأكد من ان قيمةالـ Min أقل من الـ Max .

● **Orientation** : بالرغم من انه يحمل اسم UpDown ، يمكن ان يعرض الـcomponent ازراره افقياً (udHorizontal) أو رأسياً (udVertical) . استخدم أي تحديد تريده .

● **Position** : ان خاصية الـ Position تساوي القيمة الحالية لـ UpDown . (اذاً ، لماذا لم يسموها Value؟) اقرأ هذه الخاصية في وقت التشغيل لتحديد القيمة التي تم اختيارها حالياً .

● **Thousands** : حـددها بـ True اذا كنت تريد عـلامـات ترقـيم بين مجموعات مكونة من ثلاثة خانات رقمية في الـ controls التي لها قيم Position تساوي اربعة ارقام أو أكثر .

● **Wrap** : عندما تحدد هذه الخاصية بـ False ، تتوقف الزيادة والنقصان عندما يصل الـ control الى قيمته القصوى أو الصغرى . عندما تحدد هذه الخاصية بـ True ، تقل القيمة . على سبيل المثال ، اذا كانت الـ Max تساوي 100 ، فإن محاولة زيادة الـ control ليصبح 101 تجعله يرجع الى قيمة الـ Min .

الاستجابة الى الـ events :

يمكن لازرار الـUpDown ان تستجيب لبعض الـ events ولكن في الغالب تكون الحاجة الى اثنين فقط : الـ OnChanging والـ OnClick. . ويستدعي Delphi الـ OnChanging عندما تكون قيمة الـ Position الـUpDown على وشك ان تتغير . ويتم تعريف الـ event handler procedure فيما يشبه هذا :

```
procedure TMainForm.UpDown1Changing(Sender: TObject;
  var AllowChange: Boolean);
```

كما هو معتاد، يشير الـ Sender الى الـ component الذى اطلق الـ event . فى داخل الـ event handler ، حدد الـ AllowChange بـ True لتشغيل الـ UpDown control لتغيير قيمة الـ Position . حدد الـ AllowChange بـ False لإلغاء عمل الـ cancel ، ولعدم لتغيير قيمة الـ Position . فى الغالب ، إنك سوف تستخدم برنامج الـ event handler هذا لتضع حدوداً واضحة على قيمة الـ Position ، وكذلك عندما يكون الـUpDown غير مرتبط بـ cancel آخر . على سبيل المثال ، بدلاً من إبطال الـUpDown ، قد تستخدم قيمة check Box لتحديد اذا ماكنت سوف تسمح لمستخدمين بضغط الـ cancel ، . يمكن ان تكتب برمجة الـ OnChanging كما يلى :

```
begin
 if CheckBox1.Checked
    then AllowChange := True
       else AllowChange := False;
end;
```

الـ event الثانى لـUpDown الذى ترغب فى استخدامه هو الـ OnClick . وهذا الـ event يأتى قبل الـ OnChanging . استخدم الـOnClick للاستجابه لاختيار المستخدم لازرار الاسهم التابعة للـ control . ويتم تعريف OnClick event handler procedure كما يلى :

```
procedure TMainForm.UpDown1Click (Sender: TObject;
 Button: TUDBtnType);
```

يشير الـ Sender الى أى من أزرار الـ UpDown تم إختياره فى حالة تشارك controls متعددة نفس الـ event handler . يشير الـ Button parameter الى أى من زرى السهمين قام المستخدم بضغطه عندما يكون الـ Button مساوياً لـ btNext ، يكون المستخدم قد ضغط السهم الأيمن أو العلوى ؛ عندما يكون الـ Button مساوياً لـ btPrev ، يكون المستخدم قد ضغط السهم الأيسر أو السفلى .

قد تستخدم الـ OnClick event handler لأداء بعض العمليات قبل أن تتغير قيمة الـ UpDown . تذكر أن الـ OnClick يطلق قبل الـ OnChanging ، وإنه فقط عندما يرجع الـ OnChanging الـ AllowChange مساوية لـ True تتغير قيمة الـ Position الى أعلى أو أسفل بالكمية المحددة فى النقصان .

أفكار للمستخدم الخبير

• يمكن أن يكون للـ GroupBox قائمة PopupMenu منفصلة عن الـ PopupMenu الخاصة بالـ form. قم بتعيين PopupMenu لخاصية الـ PopupMenu التابعة للـ GroupBox، بنفس الطريقة التى قمت بها بتخصيص PopupMenu للـ form. إن استخدام GroupBox متعددة فى الـ form يعد طريقة جيدة لتصميم floating pop-up menus متعددة تعتمد على أين يضغط المستخدم زر الفأرة الأيمن.

• لإنشاء أزرار تظهر وتختفى إستجابة لأوامر، أو ظروف البرنامج المتنوعة، حدد خاصية الـ Visible للزر بـ True أو False. يمكنك أن تفعل هذا لأى من الـ components التى تم شرحها فى هذا الباب، وكذا غالبية الـ Delphi components الأخرى.

• اضغط مرتين أى قيمة فى خاصية الـ Color (اضغط القيمة الموجودة على اليمين؛ وليس الاسم) لفتح نافذة الـ Color Editor. اختر أى لون، أو استخدم زر الـ Define Custom Colors لفتح جزء آخر من الـ editor الذى يسمح ببرمجة ألوان مخصصة باستخدام قيم الأزرق، والأخضر، والأحمر، أو بدلاً من قيم البريق، الإمتصاص، واللون.

• أدخل نصاً فى خاصية الـ Hint للـ SpeedButton، وحدد الـ ShowHint بـ True. يرى المستخدمون عندئذ الـ hint box صغيراً عندما يضعوا مؤشر الفأرة على الزر.

• استخدم خاصية الـ Spacing الخاصة بالـ BitBtn لتعديل كمية المسافة بين صورة الـ glyph وتعليق الزر. إن القيمة الإفتراضية حسب النظام هى 4. استخدم 1- لوضع النص فى المتنصف بين الصورة وحافة الزر. حدد الـ Spacing بـ 0 اذا لم تكن تريد مزيداً من pixels بين الصورة والنص. إن الـ SpeedButton، الذى ليس له Caption، له أيضاً خاصية Spacing يمكنك استخدامها لتعديل موضع الـ glyph.



● بالرغم من أن الـ Panels تعتبر مفيدة للغاية كـ objects بصرية خالصة، إلا إنه يمكنك إنشاء event handlers لها. على سبيل المثال، قم بإنشاء OnClick event handlers لأداء عمل عندما يضغط المستخدمون Panel object. و event آخر غاية فى الإفادة هو الـ OnResize، والذى يستخدم بشكل مثالى للـ Panels المحاذيه فى الـ client area بالنافذة عندما يقوم المستخدمون بإعادة تحديد حجم النافذة، يمكن للـ OnResize event handler الخاص بالـ Panel أن يعدل مواضع الـ controls داخل النافذة.

● إن الـ SpinButton الذى تم وصفه فى هذا الباب يستخدم object من الـTTimerSpeedButton class الغير موثقة. وبناء على الـTTimerSpeedButton class، تقوم الـ TTimerSpeedButton class بإنشاء الـ Timer الذى يحاكى تكراراً ضغطه الزر عندما تكون ضاغطاً زر الفأرة. يفعل الـ component هذا بإستدعاء الـ Click method الخاصة به، والذى يشبه الـOnClick . أنظر ملف الـ Spin.pas فى دليل الـ Source\Samples الخاص بـDelphi للحصول على الـ source code لـ TTimerSpeedButton class.

● يمكن لأى control هاو أن يجمع radio buttons، و check boxes و component objects الأخرى. على سبيل المثال، اذا اردت مجموعتى radio buttons دون حدود، استخدم الـ TPanels كحاويات وأغلق حدودها.

● استخدم الـ OnClick والـ OnChanging مع الـ UpDown عندما لا يكون مرتبط بـ control آخر. فى المعتاد، عندما يكون الـ UpDown مرتبطاً بـ text control أو edit or، تكون الـ events غير لازمة.

المشروعات التى يمكنك تجربتها

(٦-١): اكتب برنامجاً يعرض Bevel و Panel فى تشكيلات عديدة ممكنة. سوف تجد هذا نافعاً كمرشد لك لإنشاء Bevels و Panels جديدة فى التطبيقات. سوف يسمح برنامجك للمستخدمين بإدخال خصائص الـobjects ورؤية تأثيراتها فى الحال.

(٦-٢): قم بتعـديل Calc32 لاسـتـخـدام القـيم الصـحـيـحـة ٦٤ بت.
ملحوظة: استخدم نوع البيانات Int64.

(٦-٣): قم بإنشـاء glyphs للـ SpeedButtons Calc32 وبشكل عـام، رتب واجهة تطبيق المستخدم الخاصة بالآلة الحاسبة.

(٦-٤): تقنية التـحريك المقترحة فى هذا الباب. أولاً قم بجـمع ملفـات الـ Doorshut.bmp والـ Dooropen.bmp باسـتـخـدام الـ Image Editor الخـاص بـ Delphi. يجـب ان يحـتـوى الملف الناتج اربع صـور glyph منفـصلة، كلاً منهـا لهـا حـجم ١٦×١٦ pixels. قـم بتعيين الـ glyph التى تم تعديلها لـ BitBtn وقم بتشغيل البرنامج- يغلق الباب عندما تضغط الزر. قد تريد ان تبدأ مكتبة من صور الـ glyph التى تم تحريكها.

(٦-٥): اكـتب برنامج glyph catalog (Glyphlst) يعـرض كل انماط الـ glyph الموجـــودة فى دليـل الـ Images\Buttons الخـــاص بـ Delphi.

(٦-٦): اكـتب برنامـجاً يعـرض اربعـة SpeedButton توضح الحـالات الممكنة لـ glyph bitmap الخاصـة بهـا (اسـتـخـدم خـصـائص الـ AllowAllUp، و Down، و Enabled). قم بتـعـيين glyph bitmap الـSpeedButtons فى وقت التـشغيل، ربما يكون قد تم اختياره من دليل ملف.

(٦-٧): مـتـقـدم. اكـتـب floating-point calculator باسـتـخـدام الـ Calc32 كـمرشـد لك لاسـتـخـدم انواع البيـانات الـ Double (8 Byte) أو Extended (10 Byte) لتخزين القيم داخلياً.

ملخص:

● يقـدم الـ Windows ثلاثة انواع من الازرار المعـيـارية push buttons، check boxes، radio buttons. يقـوم Delphi بإحـتـواء هذه الـ controls المعيارية فى الـ Button، CheckBox، و RadioButton.

• استخدم Components array الخاص بالـ form للوصول الى جميع الـ objects- على سبيل المثال، لإبطال كل button control إلا واحد فقط فى نافذة.

• تعمل الـ CheckBoxes فى المعتاد on/off toggles، ولكن يمكن أيضاً أن يعملوا three-way switches وذلك بتحديد الـ AllowGrayed بـ True. يمكن للمستخدمين حينئذ الرجوع للمربع لتشغيل، وإبطال، وجعل علامة الصح الموجودة به رمادية.

• يمكنك تلوين CheckBox والـ RadioButton control ، ولكن بسبب حداً ما فى الـ Windows ، لا يمكنك تلوين الـ Button . للحصول على زر أكثر لوناً، استخدم الـ BitBtn .

• يمكن لـ BitBtn والـ SpeedButton ان تعرض bitmap glyphs والتى تمثل هدف الـ control ان الـ glyph هو Windows bitmap تم تقديمه كـ TBitmap، وهو فى المعتاد ١٦×١٦ pixels ذو ستة عشر لوناً. ولكن، لا يوجد حدوداً على حجم الـ glyph. ويمكن للـ glyph ان يكون لها من واحد الى اربع صور منفصلة تمثل الحالات المختلفة للزر.

• تظهر الـ SpeedButton عادة فى toolbars (انظر الباب التالى)، ولكن كما يوضح تطبيق الـ Calc32 الخاص بهذا الباب، يمكنك ايضاً استخدام الـ SpeedButton كـ objects مستقلة.

• استخدم الـ GroupBox لتجميع الـ controls المتعددة بطريقة منطقة. غالباً ما يحتوى الـ GroupBox على RadioButton ، ولكن يمكن ان يحتوى ايضاً controls اخرى يمكن للمستخدمين ضغط الـ Tab لتحريك الـ focus من احد الـ GroupBox الى آخر.

• استخدم الـ GroupBoxes كاسلوب ابسط لإنشاء مجموعات من الـ RadioButton. أضف button label فى خاصية الـ Items للـ RadioGroup.

• الـ Panel والـ Bevel عبارة عن visual component، و لكنها تفيد فى تنظيم شاشة مزدحمه ممتلئة بالازرار والـ controls الأخرى.

- ان Samples page tab على الـ VCL palette يقوم button controls إضافيين، وهما الـ SpinButton والـ SpinEdit . قم بتعـيين الـ glyphs الخاصة بك لخصائص الـ DownGlyph والـ UpGlyph الخاصة بهذه الـ objects لعرض اسهم علوية وسفلية أو objects أخرى مخصصة .

- استخدم الـ StaticText لإنشاء labels ذات window handles . يعتبر الـ StaticText نافعاً ايضاً فى إنشاء مساحات نص يمكن للمستخدمين ضغطها كالازرار .

- استـخـدم الـ UpDown بدلاً من الـ SpinButton والـ SpinEdit . لربط الاسهم العلوية السفلية بالـ label objects ، و static text ، edit objects .

يوضح الباب التالى كيفية استخدام الـ SpinButton والـPanel لإنشاء toolbars و status - عنصرى واجهة التطبيق الذان يجب ان يتوفران فى كل تطبيق Windows . وسوف نلقى نظرة . ايضاً على اختراع جديد نسبياً وهو الـCoolbars .

*****



الباب السابع

إنشاء الـ Toolbars، الـ Coolbars، والـ Status Panels

محتويات هذا الباب:

- Components
- Toolbars
- Status Panels
- Coolbars
- StatusBars

ان إحدى علامات الجودة فى برنامج الـ Windows المكتوب بشكل جيد هى الـ toolbar للـ speed button لاختيار الاوامر بالفأرة بسرعة. وتمثل الـ Toolbar buttons للفأرة ماتمثله المفاتيح الهامة للوحة المفاتيح- اختصارات للمستخدمين الذين لا يريدون إضاعة الوقت فى فتح قوائم لاختيار أوامر. يوضح هذا الباب كيفية إنشاء الـ toolbars فى تطبيقات Delphi.

ويوضح هذا الباب ايضاً كيفية إنشاء الـ Coolbars، والتى يمكن ان تحتوى على أكثر من bands التى يمكن ان تحمل Window control. على سبيل المثال، يمكن ان يعرض الـ Coolbar مجموعة من الازرار drop-down list ، check boxes، Internet hot-link buttons ، و controls اخرى. (اذا كان لديك Microsoft's Internet Explorer، فإنك تعرف الـ Coolbars- هناك واحد أعلى قمة نافذة الـ Explorer يعرض browser buttons ومنتقى عنوان الـ URL). يمكن للمستخدمين ضغط وسحب الفأرة لإعادة تنظيم لوحة الـ Coolbars.



وهناك عنصر واجهة تطبيق وهو الـ status panel ، والتى تظهر غالباً فى اسفل نافذة ما، وغالباً ما تكون مقسمة الى قطاعات عديدة. على سبيل المثال، يمكن ان تعرض status panel الخاصة بالـ text editor ارقام العمود والخط لموضع المؤشر الحالى. أو، يمكن أن تعرض الـ panels التاريخ والموعد لتذكير متابعى الساعة كما تأخروا عن الموعد النهائى. ويوضح هذا الباب طرق مختلفة لإنشاء الـstatus panel باستخدام الـ Delphi components، وكذا الـ StatusBar مع الـ Win32 .

ملحوظة: لان لـ Delphi طرق مختلفة لإنشاء، الـtoolbars والـ status panels (ولكن هناك طريقة واحدة لإنشاء الـ Coolbars)، فإن هذا الباب يستخدم المصطلحات العامة toolbar، Coolbar، status panels لهذه العناصر. والكلمات ToolBar و Coolbar و StatusBar تشير إلى الـ Win32 component الخاصة بـ Delphi .

Components:

فيما يلى بعض الـDelphi components لإنشاء الـtoolbars والـCoolbars والـ status panels .

- **Animate:** غالباً ما تحتوى الـ Coolbars على الـ Animate لتقديم بعض الصور الجرافيكية التى تم تحريكها لعرض إذ كان بدونها سيصبح عرضاً ثابتاً مملاً. ويوضح مظهر الـ Coolbars الخاص بهذا الباب كيفية إنشاء ايقونة جذابة تم تحريكها، والتى لها هدف عملى وهو جعل المستخدمين يعرفون ان البرنامج يعمل بايجابية. Palette: Win32 .

- **CoolBand:** والـ Objects الخاصة بهذا النوع من الـ components، والتى لا توجد على VCL palette ، يتم تخزينها فى الـ Coolbar ويقوم هو بالتحكم فيها. وكل CoolBand يمكن ان يحمل Coolbars آخر مثل الـ ToolBar أو الـ ComboBox . Palette : لا يوجد.

- **Coolbar:** يشبه toolbar فى النمط، ولكنه اكثر تعقيداً فى التشغيل، ويعد الـ Coolbar object متعدد القطاعات مقسم الى مساحات تسمى bands (انظر الـ



(CoolBand). ويقدم الـ Coolbar صفة الـ wrapper للـ windows CoolBar control والذى يمكنه عرض وإدارة أى عدد من الـ windowed controls (وبخاصة، تلك الـ components التى تنحدر من الـ TWinControl). يمكن للمستخدمين تشكيل الـCoolbars اثناء التشغيل بالضغط والسحب بالفأرة. Palette : Win32 .

● **Panel**: استخدم هذا الـ component لإنشاء مسطح لإقامة toolbar و status panel . يمكنك تغيير مظهر الـ 3D للسطح لنجعله يبدو مثل الفجوة المفرغة أو الهضبة المرتفعة. عند استخدام الـ panel كـ toolbar، فإنه يحمل SpeedButtons ولكن يمكن ان تعمل الـ panel كحاوية لاغراض عامة يمكنها حمل panels اخرى ونصوص لإقامة مساحات عرض قطاعية. Palette : Standard .

● **SpeedButton**: يوضح هذا الباب كيفية استخدام الـ SpeedButtons لإقامة toolbars باستخدام الـ Panel. راجع ايضاً الباب السابق لمزيد من المعلومات حول الـ events وخصائص الـ SpeedButton . الـ:Palette Additional .

● **StatusBar**: إحدى طرق إنشاء status panel هى استخدام الـ StatusBar، والـ StatusBars، والتى تظهر بالضبط فى اسفل النافذة، تحتوى واحداً أو أكثر من الـ StatusPanel . Palette : Win32 .

● **StatusPanel**: وهذا الـ component. يشكل قطاعاً، يسمى Panel، من الـ StatusBar و يتم عرضه والتعامل معه من قبل الـ StatusBar . (لاتخلط بين هذا الـ component والـ component الاكثر عموماً Panel): الـ Palette : لايوجد.

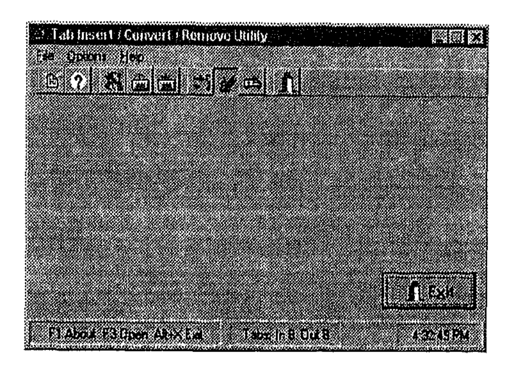
● **ToolBar**: يقدم هذا الـ component طريقة بديلة لإنشاء الـ ToolBar في نافذة تطبيق والذى يستطيع عرض الـ ToolButtons باساليب متعددة. استخدم. الـ ToolBar لتمنح تطبيقاتك مظهر يشبه متصفح Web ولتواكب إرشادات واجهة التطبيق للـ Windows 98 . Palette : Win32 .

• **ToolButton**: يمكن للـ ToolBar ان يعرض ويحتفظ بواحد أو أكثر من الـ ToolButton ، والتى تشبه الـ SpeedButton . مثل الـ SpeedButton ، يمكن للـ ToolButtons ان تعـرض ايقـونات ، ولكن يمكنهـا ايضـاً عـرض text labels . وكذلك ، يمكن ان يكون للـ ToolButtons صور مرتبطة عديدة تغير مظهرها عندما يشير المستخدم الى أو يضغط زراً . Palette : لايوجد .

الـToolbar:

On the CD-ROM

على القـرص المدمج: ان تطبيق الـ Tabs application فى هذا البـاب يوضح تقنيات الـ toolbar و status panel ، يوضح شكل (٧-١) مثال للـ toolbar for Tabs . بالرغم من ان الـ source code للبرامج طويلة جداً لكى تذكر هنا بأكملها ، توجد جميع الملفات على القرص المدمج فى دليل الـ Source\Tabs . افتح ملف مشروع Tabs.dpr لفحص ملفات التطبيق . هذا نموذج لبرنامج يستخدم units متعددة ، جميعها مرتبطة بـ visual form الا واحدة . على سبيل المثال ، ان الـ Inidata unit فى دليل الـ Source\Inidata تقدم خدمات ملف ابتدائية لتطبيق الـ Tabs .



شكل (٧-١): تطبيق الـ Tabs يحذف ويضيف tab control codes فى ملفات النص، ويمكنه ايضاً تحويل tab interval settings

يعتبر تطبيق الـ Tabs أكثر من مجرد برنامج عرض- فهو Utility مفيد قد تود ان تحتفظ به على قرصك لفترة اطول بعد الانتهاء من هذا الكتاب . استخدم الـ Tabs لإدخال وحذف tab control codes من ملفات النص ، ولتحويل الملفات

ذات tab مختلفة . على سبيل المثال ، يمكن للـ Tabs ان تحول ملفاً يستخدم tabs ذات اربع مسافات الى ملفاً يستخدم tabs معيارية ذات ثمانية مسافات . يمكن للبرنامج ايضاً ان يحول الـ Tab الى مسافات ، أو يستبدل المسافات باTabs (وهى ايضاً تقنية ضغط ملف ، رغم انها لا ينظر إليها كذلك) . لاستخدام البرنامج ، اختر الـ SpeedButtons التالية : Convert ، أو Remove ، أو Insert ، ثم اضغط زر الـOpen لفتح ملف . اجب بـ Yes لبدء الملف . استخدم اوامر الـOptions وازراره لتجهيز مواصفات البرنامج .

Caution

تحذير: تأكد من صنع بدائل لجميع الملفات قبل التشغيل مع الـ Tabs ! ان بعض المجموعات من خيارات الـ tab يمكن ان تنتج تأثيرات غير مرغوبة ، وقد تحتاج الى تجربة عدة مواصفات مختلفة لتحصل على النتيجة المنشودة . بعد تشغيل الملف ، قد يكون من الممكن استعادة تهيئة النص الاصلى لهذا الملف ، لذا تكون البدائل (أو النسخ الاحتياطية) هامة . لقد صممت البرنامج على حفظ الملفات الأصلية بعد إعادة تسميتها مع الامتداد bak ، ولكننى اقترح بشدة ضرورة ان تحتفظ بنسخ احتياطية منفصلة من ملفاتك الأصلية بالإضافة الى ذلك التى ينشئها الـ Tabs .

Note

ملحوظة: بالمناسبة ، لقد كتبت الـ Tabs اصلاً فى الـ Turbo Vision للـ DOS باستخدام الـ Turbo Pascal . ولقد استغرق تحويل البرنامج الى Delphi بضع ساعات ، وحوالى عشر دقائق لتحويله الى Delphi 3 & 4 (كانت هناك حاجة الى قليل من التغير فى الـ Inidata unit بالإضافة الى تعديلات قليلة للعرض) . ان كل برمجة واجهة التطبيق للمستخدم جديدة ، ولكن Functions و procedures إنتاج الجداول لم تتغير الى حد كبير عن الـ code الأصلية .

إنشاء الـtoolbar:

يحتوى الـ toolbar على Panel object أو أكثر من الـ SpeedButtons . لإنشاء الـtoolbar إتبع هذه الخطوات :

١ - أضف الـ Panel على الـ form . ان قيمة الـ Panel Height الإفتراضية 41 . إننى افضل 25 ، ولكن يمكنك استخدام أى قيمة تريدها .

٢- قم بتـعـيين Name مناسب للـ Panel مـثل ToolbarPanel . هذا هو الأسم الذى يستخدمه الـ Tabs .

Note

ملحـوظـة: بالرغم من ان Delphi يقـدم الـ ToolBar الخـاصـة بالـ Win32 لإنشـاء toolbars ، الا ان هذا الفـصـل يوضح كـيـفـيـة إنشـاء الـtoolbar باستـخدام انواع اخرى من الـ objects . ولان الـ ToolBars قد تم إدخالها بشكل مثالى فى الـ Coolbar ، فإننى سوف اشرح . الـ ToolBar مع المعلومات المذكورة عن الـ Coolbar فى هذا الباب .

٣- حـدد خـاصـيـة الـ ToolbarPanel's Align بـ alBottom أو alTop ، إعتماداً على ما اذا كنت تريد ان يظهر الـ toolbar فى أعلى أو اسفل النافذة . يمكنك ايضاً تصـميم toolbar تظهر الى الحـدود اليسرى (alLeft) أو اليمنى (alRight) بالرغـم من ان المكان المعـيـارى هو فى أعلى الـ client area بالنافـذة ، تحت menu bar بالنافذة مباشرة .

٤- للحـصـول على toolbar ذى صـفين أو أكـثـر من الـ SpeedButtons ، أضف panels متعـددة على الـ form وحدد خاصية الـ Align لهم بـ alTop . هذا يرتب الـ panels كالأجهزة الموضـوعة تحت الـ countertop . يمكنك عندئذ إضـافة SpeedButtons على الـ Panel .

٥- بعد إنشاء الـ Panel ، قم بحذف الـ Caption الخاص بها وادخل ماتحتاج من الـ SpeedButton . قـم دائـمـاً بإنشـاء الـ Panel أولاً ، ثـم اسـقـط الـ SpeedButtons عليهـا- هذا يجعل Delphi يعـرف ان الـ Panel هى صـاحـبة الـ SpeedButton . إننى اجـعل اسـم الـ SpeedButton الخـاصـة بى OpenSB و ConfigSB ، والذى يذكرنى بانواع الـ component types . فى الغـالب إنك تود ان تنشئ الـ OnClick لكل الأزرار ، ويمكنك التشارك فى الـ event handlers مع اوامر وازرار القائمة .

هذا هو كل ما تحتاج فعله لإنشاء toolbar من الـ SpeedButtons . بسيط ، أليس كذلك؟ .

ولكن ، هناك المزيد عن التحكم فى toolbar أكثر مما تراه العين . على سبيل المثال ، إنك غالباً تريد ان تضيف نص الـ Hint لكل زر . لتفعل هذا ، اكتب وصفاً

مختصراً في خاصية الـ Hint للـ SpeedButton's ، وحدد الـ ShowHint بـ True . يظهر عندئذ مربع معلومات صغير عندما يوضح مؤشر الفأرة على الزر لثانية واحدة . هناك سبب بسيط لعدم إضافة ملحوظات لأزرار الـ toolbar الخاصة بك- إنها تجعل البرامج ايسر في الاستخدام .

تستطيع الـ Panels ايضاً ان تستجيب للـevents، فمثلاً، يمكنك إنشاء OnClick لـ Panel object لأداء عمل عندما يضغط المستخدم، ليس على زراً ، ولكن داخل الـ Panel نفسها . ومؤخراً في هذا الباب، سوف اشرح كيفية استخدام هذه الخاصية لإنشاء floating toolbar يمكن للمستخدمين ضغطه وسحبه حول النافذة . وبوجه عام، من الأفضل عدم توجيه للـ Panels events- فإن المستخدمين يعرفون بديهياً انه يمكنهم ضغط ازرار في toolbar، ولكن توفير toolbar قابل للضغط قد يكون أكثر تضليلاً عن المساعدة . (ان الـ Coolbar يعد خياراً افضل اذا كنت تريد توفير هذا المستوى من التفاعل في واجهة تطبيق المستخدم الخاص بك . انظر الـ Coolbars في هذا الباب لمزيد من المعلومات حول إنشاء toolbar قابلة للتشكيل في وقت التشغيل) .

الـ SpeedButtons:

عند تصميم toolbar، ضع في الاعتبار اذا كنت تريد string-loaded Buttons بمجرد ان تطلق زر الفأرة، أو مجموعة من الـ objects تعمل مثل الـ radio buttons . على سبيل المثال، يمكنك إنشاء SpeedButton "sticky" تبقى مضغوطة عند دفعها؛ ويمكنك ايضاً إنشاء مجموعات SpeedButton بحيث ان ضغط زراً واحداً يؤدي الى إبطال إختيار آخر .

ان الـ SpeedButton المحملة هي الاسهل في إنشاءها . فقط أضف SpeedButton object في Panel لـ toolbar، وقم بإنشاء الـ OnClick الخاص بها . (انظر الباب السابق لمزيد من المعلومات عن برمجة الـ SpeedButtons، وكيفية التعامل مع الـ glyph Bitmap ، وإظهار صور الزر) . تعتبر spring loaded Buttons نافعة في عمليات مثل فتح ملف، طباعة، ومهام أخرى تبدأ مباشرة . بالتبادل، يمكنك إنشاء on/off SpeedButtons تبقى الى أسهل عندما تضغطها، ثم تقفز عندما تضغطها مرة أخرى .

On/off (sticky) SpeedButtons:

ان هذا النوع من الـ Toolbar Button يعتبر نافعاً للخيارات التى تختار خصائص البرنامج. فمثلاً، فى برنامج معالج كلمات قد يختار الـ SpeedButton أنماط كتابة مثل الخط السميك، المنحرف، وضع الخط تحت كلمات. قد يضغط المستخدمون الزر مرة واحدة لتحويل نص كتابة تم إختياره الى الخط السميك؛ ثم يضغطوا الزر مرة اخرى لإعادة النص الى الوضع الطبيعى.

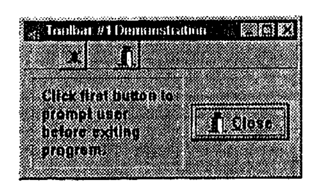
إتبع هذه الخطوات لإنشاء SpeedButtons :

١ - حدد خاصية الـ GroupIndex للـ SpeedButtons بأية قيمة غير صفر لايستخدمها أى control آخر فى toolbar .

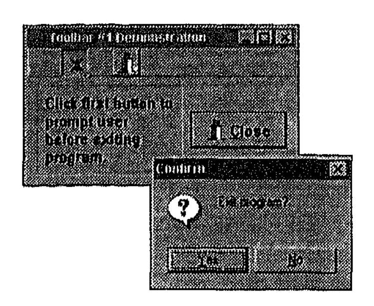
٢- حدد خاصية الـ AllowAllUp للـ SpeedButton's بـ True .

٣- باختيارات حدد خاصية الـ Down للـ SpeedButton بـ True لعرض الزر أولاً فى حالة ضغطه.

يوضح التطبيق الـ spring-loaded و sticky SpeedButtons، وتوجد ملفات البرنامج على القرص المدمج فى دليل الـ Source\Toolbar1. يوضح شكل (٧-٢) مظهر البرنامج. ان زر الـ SpeedButton الايسر sticky؛ إنه يظل الى اسفل عندما تضغطه. افعل هذا الآن واختر Close، الذى يحثك على إعطاء تأكيداً قبل إنهاء البرنامج، كما هو موضح فى شكل (٧-٣). اطلق زر الـ SpeedButton الايسر لتخرج بشكل طبيعى دون عرض message dialog. توضح القائمة (٧-١) الـ source code للبرنامج.



شكل (٧-٢): عرض برنامج الـ Toolbar1 يستخدم إثنين من ازرار الـ SpeedButtons فى toolbar بسيط. الزر الايسر sticky أى يبقى الى اسفل عندما تضغطه



شكل (٧-٣): عندما الـ SpeedButtons الواقع في أقصى اليسار الخاص بالـ toolbar مضغوطاً، يطلب Toolbar1 تأكيداً قبل الإنهاء

القائمة (٧-١):Toolbar1\Main.pas

```
unit Main;

interface

uses
  SysUtils, Windows, Messages, Classes, Graphics,
    Controls, Forms, Dialogs, StdCtrls, Buttons, ExtCtrls;

type
  TMainForm = class(TForm)
      Panel1: TPanel;
        PromptSB: TSpeedButton;
        ExitSB: TSpeedButton;
        CloseBitBtn: TBitBtn;
        Label1: TLabel;
        Bevel1: TBevel;
        procedure ExitSBClick(Sender: TObject);
        procedure FormActivate(Sender: TObject);
      private
        { Private declarations }
    public
```

```
        { Public declarations }
         end;

var
    MainForm: TMainForm;

implementation

{$R *.DFM}

procedure TMainForm.ExitSBClick(Sender: TObject);
begin
 if not PromptSB.Down then
      Close
        else if MessageDlg('Exit program?',mtConfirmation,
      [mbYes, mbNo], 0) = mrYes then
          Close;
end;

procedure TMainForm.FormActivate(Sender: TObject);
begin
 ExitSB.Glyph := CloseBitBtn.Glyph;
end;

end.
```

يوضح ملف الـ Main.pas الخاص بالـ Toolbar1 إثنين من التقنيات الهامة للعمل بالـ SpeedButtons . للزر الـ sticky يمكنك استخدام عبارة if لاستكشاف ما اذا كان الزر مضغوطاً. على سبيل المثال، انظر السطر الأول في procedure ExitSBClick، والذى ينفذ العبارة اذا لم يكن زر الـ PromptSB مضغوطاً:

```
if not promptSB.Down then ...
```

ثانياً، توضح الـ module كيفية سرقة glyph من object آخر للعرض فى واجـهـة الـ SpeedButton . لاحظ فى شكل (٧-٢) ان الـ SpeedButton على اليمين وزر الـ Close لهما نفس الـ glyph . يسرق الـ SpeedButton هذه الصورة فى وقت التشغيل بتنفيذ العبارة التالية فى OnCreate أو OnActivate :

```
ExitSB.Glyph := CloseBitBtn.Glyph;
```

مجموعات الـ SpeedButton:

ان إنشاء مجموعة من الـ SpeedButton اللاصقة والتى تعمل مثل radio button يعتبر procedure أساسى . فقط اختر كل الازرار فى المجموعة وحدد خصائص الـ GroupIndex لهم بأى قيمة غير الصفر لم يتم استخدامها من جانب أى toolbar control آخر . ومع هذا التحديد، فإن ضغط زر لم يتم اختياره يؤدى الى ظهور الزر الذى يتم اختياره حالياً مرة اخرى .

فى البداية، يمكن ان تكون جميع الازرار فى حالة الرفع . ولكن بمجرد ان يضغط المستخدم زراً، فعلى الأقل يبقى زراً واحداً الى اسفل . اذا كنت تريد ان يتمكن المستخدمون من إبطال اختيار جميع الازرار فى مجموعة، حدد خاصية الـAllowAllUp بـ True لكل زر يمكن ان يعود الى حالة الرفع . .

فى وقت التصميم، يمكنك تحديد خاصية الـ Down للـ SpeedButton بـ True لتعرضها فى البداية فى حالة إنخفاض . ولكن لكى يعمل هذا، يجب ان تحدد خاصية الـ GroupIndex للزر بأى قيمة غير الصفر . استخدم قيمة فريدة بين كل الازرار الا اذا كنت تنوى تجميعهم كما تفعل فى مجموعةالـ radio buttons .

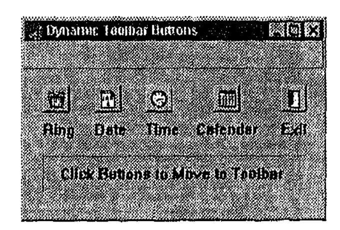
Tip فكرة: اذا رفضت خاصية الـ Down للـ SpeedButton ان تتغير الى True، حدد خاصية الـ GroupIndex للزر بأى قيمة غير مستخدمة غير الصفر . اذا ظل الـSpeedButton ملتصقاً فى الموضع السفلى- أى اذا رفضت خاصية الـ Down ان تعود الى الـ False، حدد الـ AllowAllUp بـ True . يمكنك عندئذ إعادة تحديد الـ Down ليقفز الزر .

يظهر برنامج الـ Tabs [انظر شكل (٧-١)] كيفية برمجة مجموعة من الـ SpeedButtons لكى تعمل كـ toolbar قم بتشغيل الـ Tabs واختر ازرار الـ

Insert ، الـ Remove ، والـ Convert لاختيار العملية التى تريد أدائها على ملف النص . واحداً من هذه الازرار الثلاث يكون دائماً فى حالة إنخفاض ويعمل الزران الآخران فى الـ Tabs كـ spring-loaded push buttons .

:Dynamic toolbars

باستخدامجموعة من الـ MouseDown والـ MouseUp والـ OnClick events ، يمكنك انشاء dynamic toolbars يمكن للمستخدمين تشكيلها فى وقت التشكيل . والتطبيق (Toolbar2) الموضح فى شكل (٧-٤) ، يظهر هذه التقنيات ، ويوضح ايضاً بعض طرق استغلال الـ component objects و الـ event handlers باستخدام عبارات برنامج . توجد ملفات البرنامج على القرص المدمج فى دليل Source\Toolbar2 .



شكل (٧-٤): اضغط الـ SpeedButtons على form الـ Toolbar2 لتحريها الى Toolbar يمكنك عندئذ ضغط الازرار لكى تقوم بأداء اعمالها

قم بتشغيل البرنامج واضغط الـ SpeedButton فى الـ form لنقلها الى داخل الـ toolbar بعد نقل زر الى toolbar إضغط الزر لأداء عمل مثل عرض تقويم [انظر شكل (٧-٥)] . اضغط زر الـ calendar مرة اخرى لإزالة التقويم . يجب ان تنقل الـ SpeedButton الى Exit الى toolbar قبل ان تضغطه لإنهاء البرنامج . توضح القائمة (٧-٢) الـ source code للبرنامج .

Dynamic Toolbar Buttons

Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

شكل (٧-٥): بعد نقل الـ SpeedButtons الخاصية بالـ Toolbar2 الى Toolbar، اضغط زر الـ calendar لعرض تقويم شهرى

القائمة (٧-٢): Toolbar2\Main.pas

```
unit Main;

interface

uses
  SysUtils, Windows, Messages, Classes, Graphics,
    Controls, Forms, Dialogs, ExtCtrls, StdCtrls, Buttons,
  Grids,
    Calendar;
type
    TMainForm = class(TForm)
     ToolbarPanel: TPanel;
        RingSB: TSpeedButton;
        DateSB: TSpeedButton;
        TimeSB: TSpeedButton;
        CalendarSB: TSpeedButton;
        ExitSB: TSpeedButton;
        Label1: TLabel;
        Label2: TLabel;
        Label3: TLabel;
        Label4: TLabel;
        Label5: TLabel;
        Label6: TLabel;
        Bevel1: TBevel;
        Calendar1: TCalendar;
        procedure SBMouseDown(Sender: TObject;
          Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
            procedure SBMouseUp(Sender: TObject; Button:
     TMouseButton;
          Shift: TShiftState; X, Y: Integer);
            procedure RingSBClick(Sender: TObject);
        procedure DateSBClick(Sender: TObject);
        procedure TimeSBClick(Sender: TObject);
        procedure CalendarSBClick(Sender: TObject);
        procedure ExitSBClick(Sender: TObject);
```

```
private
      { Private declarations }
        function InToolbar(Sender: TObject): Boolean;
       public
      { Public declarations }
       end;

var
    MainForm: TMainForm;

implementation

{$R *.DFM}

const
    isNotInToolbar = 0;    { SpeedButton Tag flag }
    isInToolbar    = 1;    { SpeedButton Tag flag }

{- Returns True if Sender is in the toolbar }
function TMainForm.InToolbar(Sender: TObject): Boolean;
begin
 with Sender as TSpeedButton do
    if Tag = isNotInToolbar then
    begin
     Tag := isInToolbar;  { Set Tag flag }
       Result := False;     { Return function result = False }
      end else
     Result := True;       { Return function result = True }
end;

{- Assign OnClick event handlers for buttons not in toolbar }
procedure TMainForm.SBMouseDown(Sender: TObject;
 Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
 with Sender as TSpeedButton do
    if Tag = isNotInToolbar then
```

```
    begin { Assign OnClick event handler to a button }
      if Sender = RingSB then
          RingSB.OnClick := RingSBClick
            else if Sender = DateSB then
          DateSB.OnClick := DateSBClick
            else if Sender = TimeSB then
          TimeSB.OnClick := TimeSBClick
            else if Sender = CalendarSB then
              CalendarSB.OnClick := CalendarSBClick
      else if Sender = ExitSB then
          ExitSB.OnClick := ExitSBClick;
          end;
end;

{- Move buttons into the toolbar }
procedure TMainForm.SBMouseUp(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  with Sender as TSpeedButton do
    if Tag = isNotInToolbar then
    begin  { Move button into the toolbar }
      Parent := ToolbarPanel;  { ToolBar now owns the button }
        Top := 0;              { Reposition button }
       end;
end;

{- Respond to Ring button click }
procedure TMainForm.RingSBClick(Sender: TObject);
begin
    if InToolbar(Sender) then
      MessageBeep(0);
end;

{- Respond to Date button click }
procedure TMainForm.DateSBClick(Sender: TObject);
begin
```

```
    if InToolbar(Sender) then
      ShowMessage('The date is ' + DateToStr(Date));
  end;

  {- Respond to Time button click }
  procedure TMainForm.TimeSBClick(Sender: TObject);
  begin
   if InToolbar(Sender) then
          ShowMessage('The time is ' + TimeToStr(Time));
  end;

  {- Respond to Calendar button click }
  procedure TMainForm.CalendarSBClick(Sender: TObject);
  begin
   if InToolbar(Sender) then
      with Calendar1 do
      begin
        Align := alClient;
          Visible := not Visible;
         end;
  end;

  {- Respond to Exit button click }
  procedure TMainForm.ExitSBClick(Sender: TObject);
  begin
   if InToolbar(Sender) then
        Close;
  end;

  end.
```

لإنشاء Toolbar2، قم بأداء الخطوات التالية :

١ - أضف Panel وعدد من الـ SpeedButton objects على الـ form.

٢- قم بإنشـــاء OnClick event handlers مـــثل RingSBClick و DateSBClick لكل زر.

٣- في الـ Object Inspector، قم بابراز كل OnClick event واضـغط Del لحذفه. هذا يترك الـ procedure في source module، ولكن يفصله عن الزر- وهذا ضرورياً لانه في المرة الأولى التي يقوم فيها المستخدمون بضغط الزر ينقله البرنامج الى الـ toolbar بدلاً من ان يؤدى عملاً.

للإشارة اذا ما كان هناك زراً في الـtoolbar يفحص البرنامج حقل الـ Tag الخاص بالزر، والذى يمكن ان يساوى واحد من الثابتين، وهما isNotInToolbar أو isInToolbar. يعـرف البـرنامج هـذه الثـوابت مبـاشـرة عـقب كلمـة الـ implementation في الـ unit module. وتعـــود InToolbar Function بـTrue أو False إعتماداً على قيمة الـ Tag وتغير هذه القيمة الى isInToolbar.

Tip

فكرة: يمكنك استخدام حقل الـ Tag الخاص بالـ object لأى قيمة عدد صحيح. كما هو موضح هنا، ان الاستخدام الأمثل هو عبارة عن flag يشير إلى بعض الحقائق عن حالة. الـ object..

ان كــلاً من الـ SpeedButtons يتـــشـــارك في نفس MouseDown و MouseUp event handlers. في الـ SBMouseDown procedure، تتحقق عبارة if اذا ما كان الـ Tag مساو لـ isNotInToolbar، وفي هذه الحالة يمكننا ان نفترض ان المستخدم قد ضغط زراً ينقله الى toolbar. لضمان ان الاستخدام التالى لهذا الزر ينفذ عـمـلاً، يعيـد الـ SBMouseDown إلحاق الـ OnClick الخاص بالزر، الذى كان قد تم فصله سابقاً.

عند إطلاق الفأرة، يتحقق الـ SBMouseUp procedure مرة اخرى مما اذا كان الـ Tag مساوياً لـ isNotInToolbar. فإذا كان كذلك، ينقل الـprocedure الزر الى toolbar بتحديد خاصية الـ Top للزر بـ (0)، والتى تعيد وضع الزر بالنسبة لحد الـ Panel. وتؤدى هاتان العبارتان الاعمال الضرورية:

```
Parent := ToolbarPanel;

Top := 0;
```

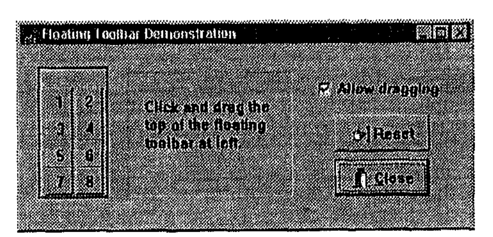
ويقوم كل OnClick event باستدعاء الـ InToolbar function لفحص ما اذا كـان الزر قـد تم نقله الى شـريط. اذا عـادت الـInToolbar بـ True، يؤدى الـ

عــــمل الزر . وبالنســـبـــه ، انظر الى كـــيف يقـــوم الـ event handler CalendarSBClick procedure باظهار وأخفاء الـ calendar object ببساطة بربط خاصية الـ Visible له بهذه العبارة :

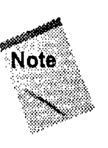
```
Visible := not Visible;
```

Floating toolbars:

من الممكن ان لا تظل الـ Toolbar panels ثابتة فى مكانها . يمكنك ايضاً إنشاء toolbars للمستخدمين ضغطها وسحبها الى أى مكان على الـ form . ان التطبيق Toolbar3 ، الموضح فى شكل (٧-٦) ، يكشف هذه التقنية المفيدة ، والتى تعتبر قيمة بشكل خاص فى البرامج ذات العروض المزدحمة ، أو أولئك الذين يعرضون صور جرافيكية كبيرة . وتوجد ملفات البرنامج على القرص المدمج فى دليل الـ Source\Toolbar3 . يستطيع المستخدمون ببساطة سحب floating toolbar بعيداً عن الشاشة بدلاً من الاضطرار الى تحريك النافذة . لتجربة هذا البرنامج ، اضغط واسحب الحد العلوى للـ toolbar . اختر أى زر ، والذى يؤكد ببساطة اختيارك . إغلق الـ check box لتثبيت الـ toolbar فى مكانه . اضغط Reset لإعادة الـ toolbar الى مكان البداية . وتوضح الـ source code فى القائمة (٧-٣) تفاصيل إنشاء الـ Panel القابل للسحب . بعد القائمة ، سوف اشرح كيف يعمل البرنامج .



شكل (٧-٦) : يوضح الـ Toolbar3 كيفية إنشاء الـ floating toolbar للـ SpeedButtons.



ملحوظة: يقدم Delphi 4 الان نوافذ قابلة للوصل ، والتى تبسط الى حد كبير التقنيات الموجودة فى هذا الفصل . ولكن لا يزال بإمكانك استخدام هذه الاساليب اذا كنت تستخدم النسخ الأولى من Delphi ،

ولكن انظر "Creating Docking Controls" فى الباب الثانى عشر لمعرفة طريقة أخـرى أنظر لإنشـاء dockable windows يمكن اسـتـخـدامـهـا كـ floating toolbars.

القائمة (٧-٣): Toolbar3\Main.pas

```
unit Main;

interface

uses
    SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics,
    Controls, Forms, Dialogs, Buttons, ExtCtrls, StdCtrls;

type
    TMainForm = class(TForm)
     FloatingToolbar: TPanel;
        SpeedButton1: TSpeedButton;
        SpeedButton2: TSpeedButton;
        SpeedButton3: TSpeedButton;
        SpeedButton4: TSpeedButton;
        SpeedButton5: TSpeedButton;
        SpeedButton6: TSpeedButton;
        SpeedButton7: TSpeedButton;
        SpeedButton8: TSpeedButton;
        BitBtn1: TBitBtn;
        Label1: TLabel;
        Bevel1: TBevel;
        AllowDraggingCB: TCheckBox;
        ResetBitBtn: TBitBtn;
        procedure FormCreate(Sender: TObject);
        procedure SpeedButton1Click(Sender: TObject);
        procedure FloatingToolbarMouseDown(Sender:
   TObject;
      Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
          procedure FormMouseUp(Sender: TObject;
            Button: TMouseButton; Shift: TShiftState; X, Y:
```

```
          Integer);
      procedure FormMouseMove(Sender: TObject;
          Shift: TShiftState; X, Y: Integer);
           procedure ResetBitBtnClick(Sender: TObject);
        procedure AllowDraggingCBClick(Sender: TObject);
       private
      { Private declarations }
        Dragging: Boolean;
        XOffset, YOffset: Integer;
        procedure MoveToolbar(X, Y: Integer);
       public
      { Public declarations }
       end;

var
    MainForm: TMainForm;

implementation

{$R *.DFM}

{- Initialize }
procedure TMainForm.FormCreate(Sender: TObject);
begin
 Dragging := False;
end;

{- Display number of selected SpeedButton in toolbar }
procedure TMainForm.SpeedButton1Click(Sender: TObject);
begin
 ShowMessage('You selected button number ' +
     IntToStr(TSpeedButton(Sender).Tag));
end;

{- Start dragging operation on clicking in toolbar }
procedure TMainForm.FloatingToolbarMouseDown(Sender: TObject;
```

```
    Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
    if AllowDraggingCB.Checked then
    begin
      Dragging := True;     { Dragging opertion in effect }
        SetCapture(Handle);  { Send all mouse messages to form }
        XOffset := X;       { Save mouse coordinates to compute }
        YOffset := Y;        { offset from top-left corner }
      end;
end;

{- End dragging operation on releasing mouse button }
procedure TMainForm.FormMouseUp(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
    if Dragging then      { Ignore if not dragging }
    begin
      MoveToolbar(X, Y);  { Move toolbar to final location }
        Dragging := False;  { End dragging operation }
        ReleaseCapture;     { Return message handling to normal
      }
       end;
end;

{- Move toolbar if dragging operation in progress }
procedure TMainForm.FormMouseMove(Sender: TObject; Shift:
    TShiftState; X, Y: Integer);
begin
    if Dragging then      { Ignore if not dragging }
      MoveToolbar(X, Y);  { Move toolbar to mouse location }
end;

{- Move the toolbar to mouse location X and Y }
procedure TMainForm.MoveToolbar(X, Y: Integer);
begin
  FloatingToolbar.Left := X - XOffset; // Adjust location of
```

```
    FloatingToolbar.Top := Y - YOffset;  // panel top-left corner
  end;

  {- Reset toolbar to startup location }
  procedure TMainForm.ResetBitBtnClick(Sender: TObject);
  begin
   with FloatingToolbar do
      begin
       Left := 24;
          Top := 24;
        end;
      AllowDraggingCB.Checked := True;
  end;

  procedure TMainForm.AllowDraggingCBClick(Sender: TObject);
  begin
   with Label1 do
       Enabled := not Enabled;
     end;

     end
```

لإنشاء floating toolbar، أضف الـ Panel على الـ form وعين له خاصية Name مناسبة. في الـ Toolbar3، قمت باستخدام اسم FloatingToolbar، ولقد غيرت عرض الـ object الى 50- بالتحديد ضعف عرض الـ SpeedButton. ان الـ Height الخاص بالـ toolbar ليس على درجة من الاهمية. إجعله طويلاً بما يكفى لان يحمل ازرارك بالإضافة الى مسافة بسيطة عند القمة. يمكن للمستخدمين ان يضغطوا ويسحبوا داخل هذه المساحة لنقل toolbar إلى مكان آخر.

ان خصائص الـ BevelOuter والـ BevelInner الخاصة بالـ toolbar كلاهما محدد بـ bvRaised، ولكن قد تريد ان تختبر انماط وتحديدات اخرى للـ Panel. ان الـ SpeedButtons الموجودة في الـ Panel تعتبر نسخة معيارية. ببساطة أضف قدر ما تحتاج من الازرار.

للتحكم فى السحب فإنك تحتاج الى ثلاث متغيرات، وهى مقررة فى قطاع الـ private للـ TMainForm . وهذه المتغيرات هى :

- **Dragging**؛ وهو متغير Boolean والذى، عندما يكون بـ True، يشير الى ان عملية سحب يتم تنفيذها .
- **Xoffset**؛ هو عدد الـ pixels الموجودة بين الحافة اليسرى للـ Panel الخاصة بـ toolbar والأحداثى X التابع للفأرة . وهذا يجعل السحب اكثر واقعية بالسماح للمستخدمين ان يضغطوا الفأرة فى أى مكان على .visible surface .
- **Yoffset**؛ عدد الـ pixels الواقعة فما بين قمة الـ Panel الخاصة بـ toolbars والإحداثى-Y التابعة للفأرة .

بالإضافة الى هذه المتغيرات، الـ class تعرف procedure خاصاً واحداً، وهو MoveToolbar . قم بتمرير إحداثيات -X و -Y الخاصة بمكان الفأرة (من خلال الـ MouseMove event، مثلاً) لتحريك الـ toolbar الى هذا المكان .

اكتب OnCreate event handler للـ form وحدد متغير الـ Dragging بـ False . هذا الذى نحتاجه عند البداية .

بعد ذلك، قم بإنشاء الـ OnMouseDown لـ FloatingToolbar . تأكد من ان تفعل هذا للـ Panel، وليس للـ form يتحقق مما اذا كان الـ Allow drugging check box بـ on - واذا لم يكن كذلك، ينتهى الـ procedure، مما يمنع بشكل فعال الـ toolbar من الانتقال .

هناك اربعة عبارات يبدئن عملية السحب (راجع الـ MouseDown فى القائمة) :

```
if AllowDraggingCB.Checked then
  begin
      Dragging := True;
        SetCapture(Handle);
        XOffset := X;
        YOffset := Y;
     end;
```

أولاً، يحدد البرنامج الـ Dragging بـ True بحيث يمكن للـ procedures الأخرى ان تكتشف ان هناك عملية سحب تتم. والـ SetCapture، وهى API Windows function، ترسل كل رسائل الفأرة الى الـ active window وبذلك، فى اثناء السحب، لا يستطيع المستخدمون استخدام الفأرة للتحول الى مهام اخرى. اخيراً، هناك تعيينين يبدءان الـ XOffset والـ YOffset لإحداثيات الفأرة، والذان متعلقان بالـ Panel الذى يحدث له الـ MouseDown event.

عندما تطلق زر الفأرة، يتولى الأمر الـ MouseUp event الخاص بالـ form. لا تقم بإنشاء هذا الـ event للـ Panel- يجب عليك ان تنشئ الـ MouseUp للـ form، بالرغم من ان عملية السحب بدأت للـ Panel. فى الواقع، عند إنشاء أى نوع من الـ objects القابلة للسحب، يجب ان يكون الـ MouseUp event للـ form، بالرغم من الحقيقية ان الـ MouseDown يكون للـ object. والسبب فى هذا هو ان الـ SetCapture، والذى يتم استدعاءه فى الـMouseDown، يؤدى الى جعل كل رسائل الفأرة التى يتم إرسالها الى active window وهذه النافذة هى الـ form- وليس الـ Panel- ولذلك، اثناء الـ SetCapture، لا تتلقى الـPanel الـ MouseUp والـ MouseMove.

ويقوم الـ MouseUp event أولاً بالتحقق مما اذا كان الـ Dragging محدد بـ True. اذا كان كذلك، فإن الـ MoveToolbar ينقل الـ toolbar الى موضعه النهائى عند احداثيات الفأرة x و y اللذان تم تعديلهما لتشغيل اللوحة. ويحدد الـ procedure بـ False، مما يلغى عملية السحب، ويستدعى الـ ReleaseCapture لإعادة رسائل الفأرة الى انماط المرور الطبيعية لها.

Tip

فكرة: دائماً اجعل استدعاء الـSetCapture مرتبط بالـ ReleaseCapture أو سيفقد مستخدمى برنامجك التحكم فى الفأرة. اذا حدث هذا، قد تفقد عملائك.

والخطوة الأخيرة فى إنشاء الـ floating toolbar هى الـ MouseMove event. الخاص بالـ form. مرة اخرى، بسبب إستدعاء الـ SetCapture، يجب ان تنشئ هذا الـhandler للـ form event، وليس الـ toolbar. ويتحقق الـ procedure مما اذا كان الـ Dragging محدد بـTrue، مما يشير الى ان عملية

الضغط والسحب تتم الآن، واذا كان الأمر كذلك، فإنه يستدع الـ MoveToolbar لتحريك الـ Panel الخاص بالـ toolbar الى مكان الفأرة. ولان الـ toolbar يمتلك الـSpeedButton objects الخاص به، فإن هذه الـ objects تتبعه آلياً- فلا يجب عليك إعادة تحديد موضعيا؛ عليك فقط ان تنقل الـ Panel الخاص بالـ toolbar.

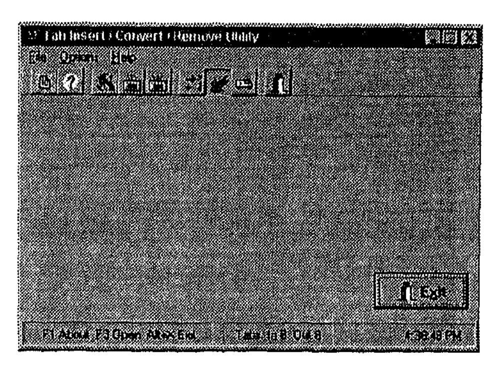
والـ procedure الأخرى فى Toolbar3 لها اهداف واضحة. يقوم الـMoveToolbar باعادة وضع الـ FloatingToolbar بتحديد قيم جديدة لخصائص الـ Left والـ Top الخاصة به. ويطرح قيم تشغيل الفأرة من الـ X والـ Y يسمح للمستخدمين ان يضغطوا الفأرة فى أى مكان من visible surface للـ Panel. (جرب تحديد الـ X و الـ Y دون طرح قيم التشغيل لترى ضرورة هذه الخطوة).

والـ ResetBitBtnClick، الذى يتم استدعاءه عندما تضغط زر الـ Reset، يعيد الـ floating toolbar الى مكان بدايته . من الصواب دائماً ان يكون لديك هذه الإمكانية لربما يفقد المستخدمون أثر toolbar الأمر الذى قد يحدث بسهولة عند إعادة تحديد حجم نافذة البرنامج.

Status Panels:

تشبه الـ status panels الـ toolbars، ولكنها غالباً ما تعرض نصاً بدلاً من الـ SpeedButton objects. وتظهر status panels دائماً فى اسفل النافذة، وقد يكون لها قطاعات فرعية تجعل قراءتها ايسر. على سبيل المثال، Microsoft Word، الذى استخدمته لكتابة هذا الكتاب، يعرض رقم الصفحة الحالية، قيم السطر والعمود، ومؤشرات لوحة المفاتيح فى Status panels مقسمة. ومن حسن الحظ انها لم تعد تعرض الوقت، والذى يذكرنى دائماً بتأخيرى فى إنهاء هذه الكتب شكل (٧-٧) يعرض status panels من خلال برنامج الـ Tabs. و توضح لوحة المفاتيح الـ functions والـAlt keys، ومواصفات الـ input and output، tab-width، والوقت الحالى. ان كل قطاع فرعى فى اللوحة يعتبر فى حد ذاته Panel object منفصل.

يمكنك ايضاً عرض online help أو رسائل خطأ فى الـ status panel. ان Delphi قد جعله من السهل إنشاء status panels.



شكل (٧-٧): برنامج الـ Tabs يعرض sectional status panel فى اسفل حد النافذة

ملحـوظة: بالرغـم من ان Delphi يقـدم الـ StatusBar component الخاص بالـ Win32 لإنشـاء status panels، فـإن هـذا الفـصـل يوضـح كيـفيـة إنشاء status panels باسـتـخدام انواع اخـرى من الـ objects . يشرح الفصل الاخير فى هذا الباب كيفية استخدام الـStatusBar component .

إنشاءstatus panel:

لإنشاء status panel، إتبع هذه الخطوات :

١- أضف الـ Panel على الـ form لتكون القاعدة لكل panels فرعى .

٢- إعط Name لهذه الـ Panel الرئيسية مثل StatusPanel، وقم بتغيير خاصية الـ Height لها لتصبح 25 . يمكنك استخدام قيمة اصغر، ولكن تذكر، ليس كل مستخدمى الحاسوب ليس لديهم ابصار حادة، ٦/ ٦، فلا تجعل لوحاتك صغيرة جداً .

٣- للحصول على تأثير 30 افضل، حدد خاصية الـ BevelInner للـ Panel بـ bvRaised، وحدد الـ BevelOuter بـ bvRaised (القيم الافتراضية) .

٤- حدد الـ BorderWidth بـ (0) (القيمة الأفتراضية) .

٥- احذف الـCaption .

٦- لتوصيل الـ Panel الى حافة زر النافذة، قم بتغيير الـ Align الى alBottom .



بالطبع يمكنك فى إختيار انماط اخرى وفى وضع الـ Panel فى أى مكان تريده .

Tip فكرة: كثير من برامج الـ Windows الناجحة تستخدم بنط الـ MS Sans Serif ذى الثمانى نقاط للـ status panel's text . مع هذا البنط ، يمكنك تعبئة حوالى ١٠٠ رمز فى status panel's خاصة بنافذة مكبرة على عرض ٦٤٠×٤٨٠ معيارى . ويبدو هذا البنط جيداً ايضاً مع الـ resolutions الأعلى .

تقسيم الـ status panel الى قطاعات:

لتقسيم status panel الى قطاعات، أضف الـ panel الإضافية فى الـ status panel (الـ Panel object الرئيسى) اختر اسماء يسهل تذكرها لكل Panel فرعية . على سبيل المثال، يسمى الـ Tabs الـ Panels الفرعية الخاصة به بـ KeyPanel، TabPanel، و TimePanel . استخدم نفس تحديدات الـ 3D للـ Panel الرئيسية، ولكن حدد الـ BevelOuter بـ None . لمحاذاة الـ Panels الفرعية تلقائياً، قم بتغيير خصائص الـ Align لها لتصبح alLeft أو alRight .

عندما تحدد الـ Align بـ alLeft أو alRight، تشترك كل panel فرعية تلقائياً مع أى panel اخرى فى statu panel الرئيسى . بعد تحديد الـ Align مبدئياً، أعد تحديد هذه الخاصية بـ alNone . يمكنك حينئذ ضغط وسحب الـ panels الفرعية الى مكان آخر - لزيادة كمية المسافة بين القطاعات، مثلاً . هذه هى الطريقة التى انشأت بها المسافات المرتفعة بين الـ panels الفرعية فى تطبيق الـ Tabs [انظر شكل (٧-٧)] .

يقدم الجدول (٧-١) خصائص هامة للـ panel objects الفرعية والـ StatusPanel فى تطبيق الـ Tabs . قد تجد هذه المعلومات مفيدة فى بناء الـ status panels المقسمة الخاصة بك .

تحديث status panels:

لتغيير النص فى الـ status panel ، ببساطة قم بتحديد خصائص الـ Caption . على سبيل المثال، لعرض الوقت، ينفذ Tabs هذه العبارة فى الـ

OnTimer الخاص بالـTimer object :

```
TimePanel.Caption := TimeToStr(Time);
```

جدول (٧-١): خصائص Panel Object Properties تطبيق Tabs

Component	Name	Property	Value
Panel	StatusPanel	Align	alBottom
		BevelInner	bvLowered
		BevelOuter	bvRaised
		Height	25
Panel	KeyPanel	Align	alLeft
		BevelInner	bvLowered
		BevelOuter	bvNone
		Font:Name	MS Sans Serif
		Font:Size	8
Panel	TabPanel	Align	alNone
Panel	TimePanel	Align	alRight

هذا يؤدى الى إن يتقدم الوقت تلقائياً. ان الـ Interval الخاص بالـ Timer محدد بـ ١٠٠٠ مللى ثانية (أى ثانية واحدة). يمكنك ايضاً تحديد Caption strings لعرض عناصر النص الثابت فى وقت التشغيل، أو يمكنك كتابة strings فى خصائص الـ Caption باستخدام الـ Object Inspector.

رغم ذلك، عليك تجنب الحاجة الى الزج بالرموز الفردية فى لوحة الـ Captions الفرعية. على سبيل المثال، من العبث ان تدخل ارقام للساعة، والدقيقة، و الثانية فى الـ Caption الخاص بالـ TimePanel- وهى محاولة اضاع فيها الكثير من واضعى البرامج فى الـ Delphi الساعات لبرمجتها. ومن الأسهل، والأكثر فعالية فى الغالب، هو ان تعيد تحديد الـ Caption strings بأكمله كل ثانية، بالرغم من ان هذا قد يبدو مضيعاً للوقت.

قد تجد انه من الضرورى، فى بعض الأحيان، ان تعرض نصاً فوق التقسيمات الفرعية للـ status panel's على سبيل المثال، يمكنك عرض ملحوظة online أو رسالة خطأ. بشكل مؤقت، إنك تريد ان تكتب فوق الـ panel's الفرعية

لتعرض الرسالة، ثم، عندما يضغط المستخدم مفتاح أو يستخدم الفأرة، يمكنك استعادة الـ status panel's الى طبيعتها.

وهذه التقنية يسهل تنفيذها. حدد الـ Alignment الخاص بالـ panel's الرئيسية بـ taLeftJustify، وقم بإخفاء الـ panel's الفرعية عندما تريد عرض الرسالة. قم بتحديد strings لخاصية الـ Caption للوحة الرئيسية. (اخيراً، اعكس هذه الخطوات لإعادة status panel الى طبيعتها.

على سبيل المثال، يعرض برنامج الـ Tabs عبارة خطأ باستخدام code شبيهة بالآتى. أولاً، يقوم البرنامج بإخفاء الـ panel's الفرعيتين اللتان تعرضان مواصفات الـ the tab-width و function key labels (ان الـ panel time الفرعية ليست عقبة، ويمكن ان تبقى مرئية):

```
KeyPanel.Visible := False;
TabPanel.Visible := False;
```

بعد ذلك، تحدد عبارة with الـ Alignment الخاص بالـ StatusPanel الرئيسية، وتحدد النص الى خاصية الـCaption:

```
with StatusPanel do
begin
  Alignment := taLeftJustify;
  Caption := 'Error #45: Now look what you''ve done!';
 end;
```

Note **ملحوظة:** اكتب علامتي تنصيص فرديتين لإدخال رمز علامة التنصيص فى string. انظر عبارة تحديد الـ Caption السابقة كمثال.

يمكنك ايضاً تحديد خاصية الـ Alignment فى وقت التصميم بالـ Object Inspector. فى هذه الحالة، يمكنك ببساطة تحديد للـ Caption string. لاستعادة الـ status panel الى وضعها الطبيعى- بعد إنتهاء الـ Timer، مثلاً، أو يتلقى الـevents الخاص بالفأرة أو لوحة المفاتيح- قم بتعيينالـ null string للـpanel الرئيسية، و حدد خصائص الـ Visible للـ panel الفرعية. على سبيل المثال، قم بتنفيذ هذا الـ code:

```
StatusPanel.Caption := '';
KeyPanel.Visible := True;
TabPanel.Visible := True;
```

استخدام الـ Format function :

تعد الـ Format function الخاصية بـ Delphi مفيدة فى إعداد strings للعرض فى status panels ، ولكن يمكنك استخدام هذه الـ utility function لعرض string-format اخرى . قد تريد مراجعة الـFormat فى online help الخاصة بـ Delphi قبل قراءة الملاحظات والاقتراحات التالية .

اذا كنت على دراية بـ C أو C++ ، فإنك ستتعرف على الـFormatعلى انها الموازى لها فى Pascal الـ sprintf() .

قم بتحديد متغيرين أو أكثر للـ Format وقم بتحديد نتيجة الـ function الى متغير string، أو قم بتمريره مباشرة إلى non-varstring parameter فى الـprocedure أو الـ function . يحدد Delphi الـ Format والـ parameters الخاصة بها كما يلى :

```
function Format(const Format: string;
    const Args: array of const): string;
```

- **Format:** متغير من نوع string أو literal يحتوى على Format specifiersتحتاجه الـ Format لإنشاء string الناتج .

- **Args:** عبارة عن open array يحتوى على واحداً أو أكثر من arguments – مثل الاعداد الصحيحة ، والمتغيرات floating-point، characters، و strings– ليتم إدخالهم فى الـ string الناتج . (من الناحية الفنية ، يمكن ان يكون لديك متغيرات اكثر من Format specifiers ولكنك تتلقى رسالة خطأ فى وقت التشغيل اذا قمت بتعريف عدد قليل) . قم بوضع المتغيرات فى الاقواس والمتغيرات المتعددة المنفصلة ضعها بين الفصلات .



ملحوظة: تسمح الـ Pascal لاسم الـprocedure أو الـ function بأن يكون هو نفس اسم الـ procedure وبذلك يكون اول للـ procedure Format مسمى بـ Format . ولكن فى الـ function، فإن تعيين قيمة

لإسم الـ function يكون امراً محيراً اذا كان procedure يستخدم نفس الأسم. فى هذه الحالة، يمكنك تعيين نتيجة للكلمة الرئيسية Result.

تعتبر الـ Format مفيدة فى إدخال string فى strings آخر. على سبيل المثال، قم بتعيين متغير string مثل التالى (اذا اردت الاستمرار، أضف Button و Label component على الـ form ثم أضف الـ code التالية فى الـ Onclick الخاص بالـ Button، قبل كلمة البداية الرئيسية):

```
var
    Proc: string;
```

قم بتعيين string لـ Proc ثم استخدم الـ Format لإدخال هذا الـ string فى آخر (ادخل هذه الـ code بين البداية والنهاية):

```
Proc := 'Pentium II';
Label1.Caption := Format('I wish my PC had a %s CPU', [Proc]);
```

يحدد السطر الأول literal string لـ Proc. يستخدم السطر الثانى الـ Format لإضافة Proc فى literal string، ابدال الـ Format specifier بـ %s بقيمة Proc's. لاحظ ان Proc's محاط بالاقواس لان المتغير الثانى الذى تم تمريره للـ Format هو open array، والذى يشبه مجموعة Pascal من حيث الاستخدام، ولكنه فى الواقع قائمة متغيرة من الـ parameters يمكن ان تكون من أى نوع. والنتيجة النهائية هى label:

```
I wish my PC had a Pentium II CPU
```

يمكنك ايضاً استخدام الـ Format لإدخال قيم عددية فى strings. على سبيل المثال، قم بتعريف متغير الـ floating-point:

```
var
    Balance: double;
```

قم بتحديد قيمة للـ Balance، وأدخلها فى string باستخدام %8.2f، والذى يحول القيمة الى string فى ثمان أعمدة ومكانين عشريين:

```
Balance := 159.72;
```

إن تنفيذ هاتان العبارتان يحدد الـ Caption الخاص بـ Label2 بالـ string التالى (لاحظ أن الأعمدة الثمانية تتضمن النقطة العشرية):



Your balance is $ 159.72

إن الـ Format specifiers يبدأ أولاً بعلامة مئوية يتبعها حرف، ويسمى النوع، وهويمثل نوع المعلومات التى يتم إدخالها فى الـ string أيضاً كيفية عرض هذه المعلومات. على سبيل المثال، تشير d% الى قيمة عشرية، و x% الى قيمة سداسية، و s% هو null-terminated string أو Pascal، و f% أو g% يشير الى قيم floating-point. إنها مسئوليتك أن توفر قيمة فى الـ Args لكل Format specifier. وإنها أيضاً مسئوليتك أن توفر قيم للأنواع المشار إليها.

فيما بين علامة الـ % وحرف النوع (d، x، s، أو أى حرف آخر) يمكن ان توجد انواع اخرى من المعلومات، مثل مؤشر الضبط الايسر (-)، قيمة عرض العمود، الدقة (يسبقها نقطة عشرية). على سبيل المثال، يهيئى الـ 6.3f% قيمة floating-point فى ستة اعمدة وثلاث اماكن عشرية. والـ 8x-% ينسق يساراً قيمة عدد صحيح سداسية فى ثمان اعمدة.

Tip **فكرة:** ان Argument index المحاط بالاقواس يبسط عملية تكرار القيم. على سبيل المثال، هذه العبارة:

```
Format ('%s %d %0:s %d', ['test',10])
```

تنتج string test 10 test 10، اذا كان لديك قيم قليلة تحتاج ان يتم وضعها فى مناطق متعددة فى string ، هذا يوفر الذاكرة التى كانت ستستهلك بسبب الـarguments المتكررة. تأكد من ان تحدد عدد الـ arguments والتى تكون كافية لكل قيمة محاطة متكررة.

ومزيداً من الامثلة تساعدك ان تستخدم وتفهم الـFormat. أولاً، حدد بعض القيم الثابتة المكتوبة (التى تشبه المتغيرات، ولكن لها أنواع بيانات معرفة وقيم إفتراضية حسب النظام):

```
const
    XInt: Integer = 123;
    XLongInt: Longint = 12345678;
    XChar: Char = '@';
    XString: string = 'My dog has knees';
```

```
XSingle: Single = 3.14159;
 XDouble: Double =  9.8695877281;
 XExtended: Extended = 9.51413;
```

أضف الـ Label على الـ form لعرض كلاً من هذه القيم . فى Button أو event handler آخر ، أدخل العبارات التالية (قد تختلف الـ Names للـ Label الخاص بك عن الخاص بى) :

```
Label3.Caption  := Format('XInt (dec) = %d', [XInt]);
Label4.Caption  := Format('XInt (hex) = %x', [XInt]);
Label5.Caption  := Format('XLongInt   = %d', [XLongInt]);
Label6.Caption  := Format('XChar      = %s', [XChar]);
Label7.Caption  := Format('XString    = %s', [XString]);
Label8.Caption  := Format('XSingle    = %f', [XSingle]);
Label9.Caption  := Format('XDouble    = %G', [XDouble]);
Label10.Caption := Format('XExtended  = %G',
[XExtended]);
```

ان تشغيل البرنامج وضغط الزر يعرض السطور التالية :

```
XInt (dec) = 123
XInt (hex) = 7B
XLongInt   = 12345678
XChar      = @
XString    = My dog has knees
XSingle    = 3.14
XDouble    = 9.8695877281
XExtended  = 9.51413
```

وايضاً أضف الـ Label لعرض نتيجة الـ string التى تم تهيئتها :

```
Pi=3.141593, Frac=0.141593, Int=3
```



على القرص المدمج: يمكن للـ Format ايضاً ان تدخل قيم متعددة فى الـ strings . إدخل ما تحتاج من الـ Format specifiers من أى نوع ، وتضع هذه الـ arguments فى اقواس وتفصل فيما بينها بالفصلات .

على سبيل المثال، القائمة (٧-٤) تحتوى على code يمكنك ادخاله فى OnClick الخـاص بالـ Button. النص مـوجـود على القـرص المدمج فى دليل الـ Source\Format فى ملف Format.pas.

ايجاد components:

استـخـدم function والـ FindComponent للبـحث عن form أو Components array لـ object آخـر بالاسم. ويرجـع هذا الـ Components array آخر، و إنه يقدم طريقة بسيطة لاداء عمليات متعددة على هذه الـ objects- إعادة تحديد حجم كل ازرار الـ form، مثلاً على toolbar أو الـ status panel.

أولا، حدد متغيراً من نوع TComponent:

```
var
    C: TComponent;
```

القائمة (٧-٤): تحديد الـ caption الخاص بالـ Label string تم تحديد مواصفاته بقيم arguments متعددة

```
procedure TForm1.Button1Click(Sender: TObject);
var
 R1, R2: Double;
    S: string;
begin
 R1 := Frac(Pi);
    R2 := Int(Pi);
    S := Format('Pi= %8.7G,  Frac=%8.6F,  Int=%G',
        [Pi, R1, R2]);
    Label1.Caption := S;
end;
```

ثم، لإيجاد component بالاسم، استخدم عبارة مثل هذه:

```
C := FindComponent('MyComponent');
```

اذا لم تكن C لاشئ، فـإنهـا ترجع الى الـ component الذى يسـمى MyComponent. اذا كـانت C لاشئ، فـلا يوجـد object يحـمل الاسم

Converted by Tiff Combine - (no stamps are applied by registered version)

المطلوب . إنك فى الغالب تتبع الاستدعاءات للـ FindComponent بعبارة if مثل هذه :

```
if C <> nil then
      { ... perform action on C here }
```

ومن الافضل ، استخدم (is) فى الـ Pascal لتأكيد ان C هو حقاً نوع الـ object الذى تعتقد انه هو الذى يجب ان يكون :

```
if C is TListBox then
   { ... perform action on ListBox object C here }
```

لاحظ ان العبارة تستخدم اسم component's class والذى يبدأ بـ T . وتقارن عبارة if بين نوع object الـ C واسم component's class . يمكنك استخدام برمجة مشابهة متى تحتاج الى تحديد نوع component's class .

فكرة: عند استخدام (is) لاختيار ما اذا كان الـ object الذى رجع بالـ FindComponent هو نوعاً معيناً ، فلا يجب عليك ايضاً ان تختبر اذا ما كانت نتيجة الـ function لاشئ . على سبيل المثال ، اذا كان التعبير "C is TListBox" حقيقياً ، إذن C لا يمكن ان تكون لاشئ .

على القرص المدمج: أنظر إلى procedure يستدعى الـ FindComponent لاستخدام الـ code ، أضف العديد من الـ check Box على الـ form بالإضافة إلى الـ Button . اضغط مرتين الـ Button واستخدم الـ procedure التالى على أنه الـ OnClick (هذا النص موجود على القرص المدمج فى دليل الـ Source\FindComp فى ملف Findcomp.pas) . قم بتشغيل البرنامج واضغط بعض الـ CheckBoxes . اضغط الـ Button لجعل كل الـ CheckBoxes off و on .

```
procedure TForm1.Button1Click(Sender: TObject);
var
      C: TComponent;
      I, J, K: Integer;
begin
      K := 0;
```



```
for I := 0 to Form1.ComponentCount - 1 do
  begin
    C := FindComponent('CheckBox' + IntToStr(I + 1));
      if C is TCheckBox then
        with C as TCheckBox do
            Checked := not Checked;
          end;
end;
```

يرجع الـ FindComponent، الـ TComponent object، وفى الغـالب ايضاً سيكون عليك ان تستخدم بعض البرمجة الإضافية لاخبار الـ Compiler أى نوع من الـ component يكون الـ C فى الحـقـيـقـة . يمكنك فـعل هذا بعـدة طرق . يمكنك استـخـدام type-cast expression من نوع الـ component وتكون الـ C بين قوسين . على سبيل المثال، هذا يحدد الـ CheckBox بـ True :

```
TCheckBoxBox(C).Checked := True;
```

أو يمكنك استـخـدام عبارة with لإخبار Delphi ان يتعامل مع الـ C كنوع معين من الـ component :

```
with Cas TCheckBox do
begin
  Checked := False;
...
end;
```

ان هذا هو المفضل بشكل عام عن الـ type casting، ان Delphi يقبل تعبير الـ type-cast دون تردد، حـتى اذا كـان C object ليس من النوع المناسب . ان استـخـدام الكلمة الاساسية as، اذا كانت C فى هذا المثال ليست object من الـ TCheckBox class، يولد Delphi خطأ الـ Invalid Type Cast .



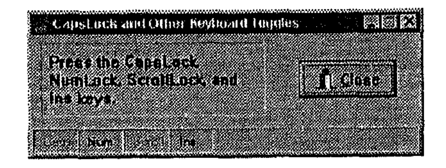
تحـذير: كن على حـذر عند اسـتـخـدام الـ FindComponent مع عـبارات with فى الـ Pascal . بالرغـم من انك سـتكون غـالبـاً تريد ان تبـحث Component array خـاص بالـ form، فـإن objects اخـرى



ايضاً لهـا هـذا الـ array وفى داخل عبـارة with ، قـد تنتـهى الى بحـث قـائمـة component آخــر . لحل هذا النوع من المشـاكل ، إمـا أن تسـتـدع الـ FindComponent داخل عـبـارة with ، أو تسـتـخـدم تعـبـيـراً مـثل MainForm.FindComponent للإشـارة الـ object و الـ Component array الذى تريد بحثه .

: Keyboard status panels

ان واحـداً من العناصـر الكلاسيكيـة للـ status panel المنظمـة هو مـجوعـة Pointers لربط لوحـة المفـاتيح . إنهـا توضـح بشكـل نموذجى تحـديدات مـفـاتيح الـ Caps Lock ، الـ Num Lock ، الـ Scroll Lock ، ومفاتيح الـ Ins . ان تطبيق الـ CapsLock ، والموضـح فى شكل (٧-٨) ، يظهـر هذا التـقنيـة . وتوجـد ملفـات البـرنامج على القـرص المدمج فى دليل الـ Source\CapsLock . توضـح القـائمـة (٧-٥) الـ source code للـ Main.pas الخاص بالبـرنامج . هناك اساليب عـديدة ممكنة لإنشاء Pointers لربط لوحـة المفاتيح ، ولكـن هذا الاسلوب الذى اعـتـقـد انه الافضل يظهر نصاً معتماً للمفاتيح فى حالة الإغلاق ونصاً مظلماً(طبيعى) للمفاتيح فى حـالة الفـتح . ولكـن ، تطبـيق هذا مع الـ TPanel objects لا يكون مـبـاشـراً بالضبط ، كما سأوضح بعد القائمة .



شكل (٧-٨): تطبيق الـ CapsLock يعرض حالات الفتح والإغلاق لمفاتيح الـ CapsLock ، الـ Num Lock ، الـ Scroll Lock ، والـ Ins

القائمة (٧-٥): Capslock\Main.pas

```
unit Main;

interface

uses
```



```
    SysUtils, Windows, Messages, Classes, Graphics,
     Controls, Forms, Dialogs, ExtCtrls, StdCtrls, Buttons;

type
    TMainForm = class(TForm)
      StatusPanel: TPanel;
        CapsLockPanel: TPanel;
        NumLockPanel: TPanel;
        ScrollLockPanel: TPanel;
        InsPanel: TPanel;
        BitBtn1: TBitBtn;
        Label1: TLabel;
        Bevel1: TBevel;
        CapsLockLabel: TLabel;
        NumLockLabel: TLabel;
        ScrollLockLabel: TLabel;
        InsLabel: TLabel;
        procedure FormKeyDown(Sender: TObject; var Key: Word;
          Shift: TShiftState);
            procedure FormActivate(Sender: TObject);
       private
      { Private declarations }
       public
      { Public declarations }
        procedure UpdateKeyPanel;
       end;

var
    MainForm: TMainForm;

implementation

{$R *.DFM}
```

```
procedure TMainForm.UpdateKeyPanel;
begin
    CapsLockLabel.Enabled :=
     GetKeyState(VK_CAPITAL) and 1 = 1;
      NumLockLabel.Enabled :=
       GetKeyState(VK_NUMLOCK) and 1 = 1;
    ScrollLockLabel.Enabled :=
     GetKeyState(VK_SCROLL)  and 1 = 1;
      InsLabel.Enabled :=
     GetKeyState(VK_INSERT)  and 1 = 1;
end;

procedure TMainForm.FormKeyDown(Sender: TObject; var Key: Word;
    Shift: TShiftState);
begin
    UpdateKeyPanel;
end;

procedure TMainForm.FormActivate(Sender: TObject);
begin
  UpdateKeyPanel;
end;

end.
```

لإنشاء status panel مرتبطة بلوحة المفاتيح، اتبع الخطوات التالية:

١- أضف الـ TPanel object على الـ form وقم بتعديله كما هو موضح في هذا الباب، على سبيل المثال، حدد الـ Height بـ ٢٥ واختر اسلوب الـ 3D.

٢- قم بمحاذة الـ panel بأسفل النافذة.

٣- أضف الـ subpanels في الـ TPanel الرئيسية، واجعل اسماءها CapsLockPanel، NumLockPanel، الى آخره.

٤- احذف كل الـ Captions الخاصة بالـ panel.

ان عـرض dim text فى الـ TPanel object [انظر شكل (٧-٨)] يعـد اقل من غير المعقد لان هذا الـ component لا يعتم نص الـ Caption الخاص به عندما تكون خاصية الـ Enabled للـ TPanel محددة بـ False. وبسبب هذا الحد، يدخل تطبيـق الـ CapsLock اربعـة Label objects فى كل subpanel. لعـرض dim label لمفتاح يكون فى حالة إغلاق، يحدد البرنامج خاصية الـ Enabled للـ Label بـ False؛ لعـرض normal label لمفـتـاح فى حـالة فـتح، يحـدد البـرنامج الـ Enabled بـ True. لا يقـوم البـرنامج بتـعـيين نص للـ Captions الخـاصـة بالـ subpanel.

يوضح الـ UpdateKeyPanel Procedure كـيـفـيـة الحـصـول على المواصـفـات الحـالية لمفاتيح الـ Caps Lock، Num Lock، Scroll Lock، و Ins. يسـتـدعى الـ procedure و Windows API function وهى GetKeyState، والتى يمكنك ان تمرر إليـهـا أى virtual key code ظاهرى ونتيجة تنفيذ الـ GetKeyState عدد صحيح. اذا كـان اقل بت فى هذه القيمة يساوى 1، يكون المفتاح مفتوحاً؛ وان لم يكن كذلك، يكون المفتاح مغلقاً. ويختبر تعبير AND المنطقى هذه البت، معطياً النتيجة فى قيمة Boolean True أو False. ان تعيين هذه القيمة لخاصية الـ Enabled للـ Label يعرضها بنمط عادى أو معتم.

لعـرض قـيم المفـاتيح بشكل سليم عندمـا يبـدأ البـرنامج وعندمـا يضـغط المستخدمون واحداً من الـ labeled keys، يقوم الـ CapsLock بإنشاء إثنين من الـ event handler للـ form وهمـا OnActivate و OnKeyDown. وكـلا من هذين البرنامجين يستدعى UpdateKeyPanel procedure لتحديث الـ status panel.

Tip

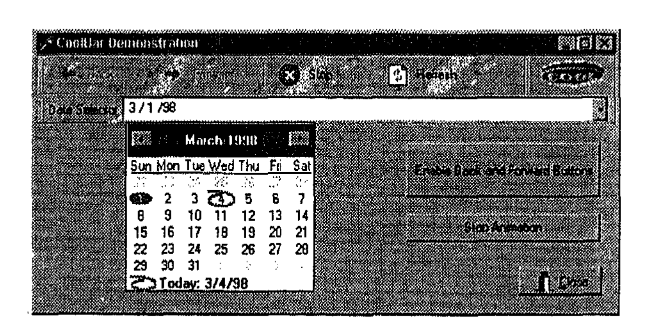
فكرة: لفـحص تغـيـرات مـفـاتيح الـ Caps Lock، Num Lock، Scroll Lock، و Ins فى OnKeyDown الخـاص بالـ form، حـدد خاصية الـ KeyPreview للـ form بـ True.

الـCoolbars :

الـ Coolbars ، والذى يعـــرف ايضـــاً بالـ rebar ، هو objects . بإمكانه إحـتـــواء أكـثـــر من objects أخـــرى . وتلك الـ objects ، والتى يمكن أن تكون objects من classes مشتقة من الـ TWinControl ، تستقر على واحدة أو أكثر من الـ bands والتى تقسم الـ Coolbar الى قطاعـات . وكل bands فى حـد ذاته يعد object من الـ TCoolBand class . فى وقت التشغيل ، يمكن للمستخدمين ضـغط وسـحب الـ bands لخلطهـا وتجـميعهـا لجعل عـرض الـ Coolbar يتـفق مع رغبـتـهـم . اثناء تصـمـيم البـرنامج ، يمكنك فـعل نفس الشئ لترتيب الـ Coolbar bands وفق رغبتك .



على القرص المدمج: يوضح شكل (٧-٩) مثال Coolbar من برنامج العرض الخاص بهذا الفصل ، وهو CoolDemo ، الموجودة فى الموجود فى دليل Source\Cooldemo على القـــرص المدمج . ومـــثـــال الـCoolbar له two bands ، واحداً فوق الآخر . ويحمل الـ band العلوى اربعة ازرار ، والذين لا يؤدون أيه اعمال حقيقة فى العرض- إنهم للعرض فقط . ويحمل الـ band الثانى DateTimePicker object ، ويعرض ايضاً "Date selector." label .



شكل (٧-٩): يوضح تطبيق الـCoolDemo كيفية إنشاء واستخدام الـ Coolbar، والذى فى هذه الحالة، يحمل اربعة ازار و DateTimePicker object

سوف تصبح الـ Coolbars أكثر إنتشاراً عندما يحصل الـ Windows 98 على القبول اللازم . اذا كنت تراجع تطبيق Delphi موجود بالفعل ، فقد تريد

التـفكير في ابدال الـ toolbars الخاصة بك بـ Coolbars . وهذا يعطي برامجك مظهراً ثابتاً بين برمجيات الـ Windows الأخرى . يوضح هذا الفصل، كيفية تقسيمه الى bands، كيفية إضافة انواع متنوعة من components الى تلك الـ bands .



ملحوظة: للحصول على مثال آخر عن الـ Coolbar، انظر برنامج متصفح Web في دليل Demo\Coolstuff الخاص بـ Delphi .

إنشاء الـ Coolbars :

ان إنشاء الـ Coolbar هو أمر غاية في السهولة- فقط أضف الـ component الـ Coolbar من Win32 VCL paletteعلى الـ from حسب النظام الافتراضي، يلصق الـ Coolbar نفسه على قمة النافذة. يمكنك تغيير هذه القيمة المتغييرة باختيار قيمة مختلفة لخاصيةالـ Align للـ Coolbar، ولكن في معظم الحالات، يكون قمة النافذة، تحت menu bar مباشرة ان وجد، وهي المكان الذي يجب عرض الـ Coolbar فيه، كما هو موضح في شكل (٧-٩).

على القرص المدمج: توضح القائمة (٧-٦) للـ source code CoolDemo في ملف Main.pas، الموجود في دليل Source\CoolDemo على القرص المدمج. سوف اشرح البرمجة الموجودة في هذه القائمة في الاماكن المناسبة في هذا الفصل والفصول التالية.

القائمة (٧-٦): CoolDemo\Main.pas

```
unit Main;

interface

uses
    Windows, Messages, SysUtils, Classes, Graphics,
  Controls,
    Forms, Dialogs, ComCtrls, ToolWin, Images, StdCtrls, Buttons;
```



```
type
    TMainForm = class(TForm)
     Coolbar1: TCoolbar;
        ToolBar1: TToolBar;
        NavigatorImages: TImageList;
        NavigatorHotImages: TImageList;
        DateTimePicker1: TDateTimePicker;
        ToolButton1: TToolButton;
        ToolButton2: TToolButton;
        ToolButton3: TToolButton;
        ToolButton4: TToolButton;
        ToolButton5: TToolButton;
        BitBtn1: TBitBtn;
        BitBtn2: TBitBtn;
        Animate1: TAnimate;
        Button1: TButton;
        procedure BitBtn1Click(Sender: TObject);
        procedure Button1Click(Sender: TObject);
      private
     { Private declarations }
      public
     { Public declarations }
      end;

var
    MainForm: TMainForm;

implementation

{$R *.DFM}

procedure TMainForm.BitBtn1Click(Sender: TObject);
var
```

```
    TF: Boolean;  // True or False flag
     S: String;
begin
    TF := ToolButton1.Enabled;
    ToolButton1.Enabled := not TF;
    ToolButton2.Enabled := not TF;
    if TF
        then S := 'Enable'
        else S := 'Disable';
    BitBtn1.Caption := S + ' Back and Forward Buttons';
end;

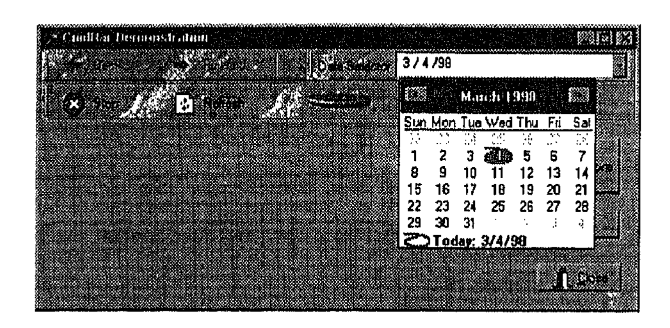
procedure TMainForm.Button1Click(Sender: TObject);
begin
    Animate1.Active := not Animate1.Active;
    if Animate1.Active
      then Button1.Caption := 'Stop Animation'
        else Button1.Caption := 'Begin Animation';
end;

end.
```

حتى تعتاد على الـ Coolbar، قم بتشغيل الـ CoolDemo الآن بتحميل ملف مشروع الـ Cooldemo.dpr فى Delphi من دليل Source\Cooldemos، واضغط F9. اضغط الزر الكبير لتشغيل وابطال الزرين الأولين فى الـ bands العلوى للـ Coolbar. لاحظ انه عندما تحرك مؤشر الفأرة فوق الزر، فيبدو وكأنه يكبر عن الشاشة وتتغير ايقونته من الاسود والابيض الى صورة ملونة. إنك توفر هذه الصور باستخدام واحداً أو أكثر من ImageList objects ، كما اوضحت قبل ذلك فى هذا الفصل.

لإعادة تشكيل تنظيم الـ Coolbar، اضغط واسحب واحداً من اثنان من الـbands الرأسيين (إنها يبدوا ان لى كالمسمارين المستدرين) فى اقصى اليسار فى الـ Coolbar band. على سبيل المثال، التقط الـ band السفلى وحركه تجاه القمة-

هذا يجمع الـ bands ويجعلها يجلسان جنباً الى جنب كما هو موضح فى شكل (٧-١٠). "قد يكون عليك ان تجرب عدة مرات لتحصل على التأثير الذى تريده).



شكل (٧-١٠): اضغط واسحب bands الرأسى لـ Coolbar band لإعادة ترتيب تنظيم الـ Coolbar كما هو موضح هنا باستخدام تطبيق الـ CoolDemo

لكل band فى الـ Coolbar لديها text label . لإنشاء label ، اضغط الزر البيضاوى الواقع بعد خاصية الـ Coolbar object Bands . وهذا يفتح الـCoolbar Bands Editor . اختر أى Band (أو أضف واحداً اذا لزم الأمر بضغط زر الـ الخاص Add New)، ثم ادخل الـ label الخاص بك فى خاصية الـ Text باستخدام الـObject Inspector التابع بـ Delphi .

ملحوظة: الطريقة الوحيدة للتوصل الى band objects فردية هى من خلال الـ Coolbar Bands Editor . بالرغم من ان كل band object من الـ TCoolBand class، الا ان objects غير مقررة بصورة فردية فى الـ form class ولذلك، فإنها غير واردة فى قائمة لائحة الـ Object Inspector .

خصائص الـCoolbar:

فيما يلى بعض الملاحظات حول خصائص الـCoolbar . المختارة. قد تحتاج الى قراءة هذا الفصل والفصول القليلة التالية قبل ان تفهم هدف بعض هذه الخصائص:

• **Align:** فى المعتاد، تستقل الـCoolbar على قمة النافذة، ولكن يمكنك تحديد الـ Align بـ alBottom، alRight ، أو alLeft لتغيير التحديد الافتراضى بـ alTop . وقد تحدد بعض التطبيقات هذه الخاصة بـ alNone للحصول على

Coolbar ذى حجم ثابت، أو alClient لملئ النافذة بخلفية الـ Coolbar . عندما تكون الـ Align بـ alTop أو alBottom، تكون الـ Vertical محددة بـ False. عندما تكون الـ Align بـ alLeft أو alRight، تكون الـ Vertical محددة بـ True. كن على حذر من هذا التفاعل فيما بين الحقلين.

• **AutoSize**: انك دائماً تحدد هذه الخاصية بـ True، وهى قيمتها الافتراضية، حيث يقوم الـ Coolbar تلقائياً بتعديل حجمه ليتلائم داخل حدود النافذة. ولكن يمكنك ان تحدد هذه الخاصية بـ False اذا كنت تريد ان تضع الـ Coolbar يدوياً.

• **Bands**: اضغط مرتين هذه القيمة الـ Coolbar Bands Editor . إستخدم الـEditor لإضافة، حذف، إعادة ترتيب الـ Coolbar Band، الذى يعتبر كلاً منهما object من الـ TCoolBand class. يعتبر الـ Bands Editor هو الطريق الوحيد الذى يمكنك بواسطته التوصل الى Bands منفردة فى وقت التصميم. اختر Band باستخدام الـ Editor ثم ادخل قيم الخاصية فى نافذة الـ Object Inspector كما تفعل مع الـ components الأخرى.

• **Bitmap**: قم بتعيين أى ملف bitmap لهذه الخاصية لتلوين خلفية الـ Coolbar. من الناحية النموذجية، يجب ان تكون الـ bitmap متماثلة، بحيث عندما تتكرر فوق الـ Coolbar، يكون التأثير البصرى الأخير بلا جوانب. من الأفضل ان تستخدم صور ذات الوان فاتحة أو رمادية فاتحة- لان الالوان الداكنه تجعل من الصعب رؤية النص والازرار الموجودة على الـ Coolbar Bands. اذا لم تكن تريد استخدام bitmap image لخلفية الـ Coolbar، يمكنك تحديد خاصية الـColor لتنظيم العرض وإذا لم تحدد Bitmap أو قيمة Color، فإن الـCoolbar له نفس لون واجهة الزر العادى الـ(Color = clBtnFace).

• **EdgeBorders**: وهذه الخاصية لها اربع قيم فرعية التى يمكنك تحديدها بالضغط مرتين على علامة الزائد الصغيرة الى اليسار من الـ EdgeBorders. فى نافذة الـ Object Inspector. ولكل قيمة تحدد بـ True أو False- قم بتجربة مجموعات متنوعة لترى تأثيراتها. والقيم الاربع الفرعية هى: ebLeft، ebTop، ebRight، ebBottom لإنشاء Coolbar ذى حدود غير مرئية، حدد القيم الفرعية الأربع بـ False، أمحو خاصية الـ Bitmap، وحدد الـ Color بـ



clBtnFace. سيكون للـ Coolbar الناتج غير مرئياً في وقت التشغيل، ولكنه يعمل وكأنه واحداً ذا حدوداً مرئية.

- **ShowText**: لعرض Labels في الـ Coolbar's bands، اختر bands باستخدام الـ Bands Editor وادخل الـ label الخاص بك في خاصية الـ Text باستخدام الـ Object Inspector. حدد خاصية الـ ShowText للـ Coolbar بـ True لعرض الـ label band. وتعرض هذه ايضاً في الـ Bands Editor .

- **Vertical**: تؤثر هذه الخاصية على كيفية عرض الـ bands في الـ Coolbar. حدد الـ Vertical بـ False لعرض الـ bands افقياً، واحداً فوق الآخر (وهو التحديد الافتراضي . حدد الـ Vertical بـ True لعرض الـ bands رأسياً، جنباً الى جنب . حدد خاصية الـAlign للـ Coolbar، والتي تؤثر على موقع النافذة، قبل تغيير الـ Vertical . يعتبر هذا ضرورياً لان تعيين قيم خاصية الـ Align قد يغير قيمةالـ Vertical . (راجع الـ Align).

Tip **فكرة**: لان الـ Coolbar غالباً ما يحمل bands لـcontrols اخرى عديدة، فإنه من الصعب عادة ان تختار الـ Coolbar object بالفأرة في Delphi. اذا حدث هذا، استخدم قائمة لائحة الـ Object Inspector لاختيار الـ object- مثلاً، افتح تطبيق الـ CoolDemo واختر Coolbar1 لفحص قيم بخاصية الـ Coolbar الخاص بالعرض.

خصائص الـ CoolBar:

فيما يلي ملاحظات حول خصائص CoolBand، وتذكر ان الطريقة الوحيدة للوصول إلى الـ band objects منفردة هي ضغط الزر البيضوي الواقع بعد خاصية الـ Bands للـ Coolbar، وان تفتح الـ Bands Editor. يمكنك عندئذ ابراز Band منفرد واستخدام الـ Object Inspector لتحديد خصائصه. ولان الـCoolbar يملك الـ CoolBand objects، فإنها لا تظهر في قائمة لائحة الـcomponents الخاصة بالـ Object Inspector .

- **Bitmap**: كل Band من الممكن أن يملك Bitmap image خاصه به للعرض في خلفيتها. اذا قمت بتعيين ملف Bitmap لهذه الخاصية، فإنها تطغى

على أى قيمة Bitmap معينة للـ Coolbar . بدلاً من استخدام الـ Bitmap ، يمكنك ايضاً تغيير خاصية الـ Color لأى Band فردى .

● **Break**: حدد هذه الخاصية بـTrue لوضع الـ band على السطر جديد على الجانب الايسر من الـCoolbar عندما تحدد الـBreak بـ False ، يتم عرض الـ band بعد أى band سابق ولا يبدأ سطر جديد .

● **Control**: يمكن ان يحمل كل band ، Control واحد . فى العادة ، يتم تحديد هذه الخاصية تلقائياً للـcomponent الذى تسقطه على الـCoolbar ، ولكن يمكنك تحديد الـ Control لأى classobject مشتقة من الـ TWinControl .

● **FixedBackground**: فى المعتاد ، حدد هذه الخاصية بـ True بحيث تكون الـBitmap التابعة للـ Coolbar معروضة باستمرار خلف كل الـ bands . عندما تكون الـ FixedBackground محددة بـ False ، تكون الـ Bitmap التابعة للـ Coolbar معروضه بشكل منفصل لكل bands وإعتماداً على نوع الـ Bitmap التى تستخدمها ، قد يكون هذا نافعاً . ولكن للحصول على مظهر بلا جوانب ، يجب ان تكون الـ FixedBackground محددة بـ True . انظر ايضاً خاصية الـ ParentBitmap .

● **FixedSize**: عندما تكون هذه الخاصية محددة بـ True لا يستطيع المستخدمون إعادة تحديد حجم الـ bands . فى العادى ، تكون هذه الخاصية بـFalse (وهو التحديد الافتراضى) .

● **HorizontalOnly**: فى الحالات التى يجب فيها عرض band معين افقياً فقط ToolBar ، على سبيل المثال- حدد هذه الخاصية بـ True . اذا كانت خاصية الـ Vertical للـ Coolbar الاكبر محددة ايضاً بـ True ، سيصبح الـ bands غير مرئياً . قد تستخدم هذه الخاصية لإخفاء bands مختارة عندما تكون جميع الـ bands مرتبة رأسياً ، ولكن من الصعب تخيل الاستخدام العملى لهذا التحديد الغامض . لغالبية التطبيقات ، استخدم التحديد الافتراضى بـ False .

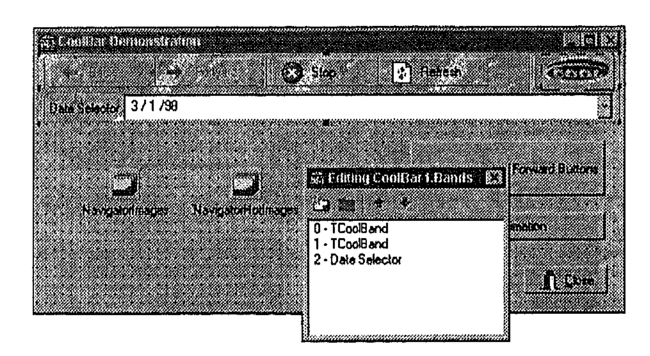
● **Image Index**: قد يعرض الـ band صورة ايقونة بتحديد هذه الخاصية بقيمة فهرس الصورة فى الـ ImageList object الخاص بها . يجب ان يتم تعيين object من لخاصية Images للـCoolbar .

● **ParentBitmap**: عندما تحدد هذه الخاصية بـ True ، ويعرض الـ bands نفس الـ Bitmap image المعينة للـ Coolbar . bands parent . ان تعيين صورة لخاصية الـ band's Bitmap يحدد تلقائياً الـ ParentBitmap بـ False .

إضافة ToolBars للـCoolbars :

بعد اسقاط الـ Coolbars على الـ form سوف تريد ان تقسمه الى bands . يمكنك ان تفعل هذا بطريقتين :

● اضغط الزر البيضاوى الواقع بعد خاصية الـ Bands للـ Coolbar فى نافذة الـ Object Inspector . هذا يؤدى الى فتح الـ Coolbar Bands Editor [انظر شكل (٧-١١)]، والذى يمكنك استخدامه لإضافة، وحذف، وإعادة تنظيم الـBands .



شكل (٧-١١): استخدم الـCoolbar Bands Editor لإنشاء، حذف وإعادة ترتيب الـbandفى الـ Coolbar

● قم باسقاط الـ windowed control مثل الـ ComboBox، ToolBar ، أو مثل ما هو فى برنامج العرض، DateTimePicker فى الـ Coolbar . وهذا ينشئ تلقائياً band جديد لحمل الـ control .

ان غالبية الـ Coolbar تعرض زراً واحداً أو أكثر لإنشاء ToolBar . لفعل هذا، قم أولاً باسقاط الـ Coolbars على الـ form ، ثم اسقط ToolBar object (موجود ايضاً على Win32 palette) على الـ Coolbar . اضغط الـ ToolBar باستخدام زر الفأرة الأيمن، واختر أمر الـ New Button لإنشاء ازرار ToolBar يعتبر كل زر object من الـ TToolButton class (وهذه ليست على الـ VCL

palette- يجب ان تســـتـخــدم امــر الـ New Button لإنشـــاء هذه الانـواع من الازرار).

Tip **فكرة:** للحصـول عـلى مظهر ToolBar كـلاسيكى، اتبع هذه الافكار: حـدد خاصية الـ AutoSize للـ ToolBar بـ True، وحـدد خاصية الـ AutoSize لكل الـ ToolButton بـ False. حـدد خـاصـيـة الـ Flat للـ ToolBar بـ True. لتعديل احجام الزر، اختر الـ ToolBar وعين قيم لخصائص الـ ButtonHeight والـ ButtonWidth. يجب ان تكـون جـمـيع الازرار فى الـ ToolBar ذات حـجم واحـد، والذى يتم تحـديده بواسطة الـ ToolBar، وليس بالازرار المنفردة، كما هو الحال فى اغلب Delphi components الأخرى.

تبـدوالـ Coolbars أكـثـر تميزاً بالخلفـيـة التى تشبـه صـورة حـجـرية فى الـ CoolDemo [راجع شكل (٧-٩)]. وهـذا هو الافـضـل مع الـ ToolBar التى تم تحديد خاصية الـ Flat لها بـ True، الأمر الذى يجعل الخلفية لامعة خلال ازرارالـ ToolBar. لانشـاء التأثيـر البصـرى فى الـ CoolDemo، قـمـت بتعـيين خـاصـيـة الـBitmap للـ Coolbar للصــورة الموجــودة على القـــرص المدمج فى صــورة Source\Data\Background.bmp. يمكـنـك اســـتــــخـــدام أى Bitmap بالـCoolbar، ولكن المثال مصـمم خصـيصـاً لتكرار أى خلفـيـة بلا جوانب من أى حجم.

والمرة الاولى التى تضيـف فيـهـا ازرار للـ ToolBar، سـوف تكون خـاوية. يوضح فصل "الصـور والصور الهامة" فى هذا الباب كيفية إضافة ايقونة لازرار الـ ToolBar. لعرض text label- والذى يمكنك فعله ايضاً بعرض ايقونات أو بدون عـرضـهـا- اخـتـر ToolButton، ادخل الـ object الخـاص بك فى خـاصـيـة الـ Caption باستخدام الـ Object Inspector. لكى تجعل الـ Caption مرئياً، اختر الـ ToolBar وحدد خاصية الـ ShowCaptions بـ True.



ملحوظة: يمكنك إضافة الـ ToolBar مباشرة على الـform- لا يجب ان يستقر الـ ToolBar على Coolbar band. ان تقنيات البرمجة هى نفس تقنيـات الـToolBar المسـتـقلة كـمـا هو مـوضح هنا لتطبـيـق الـ CoolDemo.

خصائص الـ Toolbar:

فيما يلي ملاحظات حول الخصائص المختارة للـ Toolbar. تذكر، بالرغم من ان هذا الفصل يشرح الـ ToolBars بارتباطها بالـ Coolbars، يمكنك إضافة الـ ToolBar مستقلة على الـ form. لا يجب عليها ان تبقى داخل الـ Coolbars.

- **Align**: ان اغلب الـ Toolbars تعرض على قيمة النافذة، ولكن يمكنك تغيير هذه الخاصية لمحاذاة القضيب على الحد السفلى للنافذة (alBottom)، أو الحافة اليسرى(alLeft) ، أو(alRight) . للحصول على toolbar أو floating أو ذو حجم ثابت، حدد الـ Align بـ alNone حددها بـ alClient للئ الـ client area بالنافذة- وهى تقنية جيدة لواجهات التطبيق التى تزدحم بمجموعات الازرار، مثل نظام الادخال المنظم لشاشة اللمس.

- **AutoSize**: عندما تحدد هذه الخاصية بـ True، يعدل الـ Toolbar حجمه تلقائياً ليدخل ازراره دون مسافة مهدرة. إنك فى الغالب تريد تغيير هذه القيمة الى True (فهى False حسب النظام الافتراضى) فى غالبية التطبيقات.

- **FuttonHeight, ButtonWidth**:عين قيم لهاتين الخاصيتين قبل إنشاء ازرار الـ ToolBars. جميع الازرار فى الـ ToolBars لها نفس الحجم. يتم التعبير عن القيم بالـ pixels.

- **DisabledImages**: قم بتعيين ImageList لهذه الخاصية، وعين قيم ImageIndex لكل. من ازرار الـ ToolBars. تعرض الازرار التى تم ابطالها الايقونات من الـ ImageList object هذا. اذا لم تعين ImageList لهذه الخاصية، يتم إعتام ايقونات الزر المبطل تلقائياً باستخدام Bitmap من خاصية الـImages (اذا كانت معينة).

- **Flat**:للحصول على مظهر كلاسيكى يشبه متصفح Web، وبخاصة عند إدخال ToolBar فى Coolbar ، فإنك تريد ان تحدد هذه الخاصية بـ True. قم ايضاً بتعيين الـ Bitmap للـ Coolbar ، وحدد خاصية الـ Transparent للـ ToolBar بـ True. يأخذ الـ ToolBar عندئذ مظهر الـ Coolbar ، كما تفعل ازار الـ ToolBar.



- **HotImages:** قم بتـعـيين ImageList لهـذه الخـاصـيـة، وعين قـيم ImageIndex لكل من ازرار الـ ToolBar. يتم عـرض هذه الايقـونات عندما يقوم المستخدم بتحريك مؤشرة الفأرة فوق الازرار. وبالطبع، ان الايقونات هي نفسها التي في خاصية الـ Images، ولكنها مختلفة عندما تشير إليها مؤشرة الفأرة. ان الصـور الهـامـة لا تسـتـخـدم للازرار التي تم ابطالهـا (تلك التي لهـا خـاصـيـة الـ Enabled محدد بـ False).

- **Images:** قم بتعيين ImageList لهذه الخاصية، وعين قيم ImageIndex لكل من ازرار الـ ToolBar. يتم عـرض هذه الايقـونات على الازرار الا اذا كـانت مـبطلة أو كـان المسـتـخـدم يشـيـر الى زر باسـتـخـدام الفـأرة (راجع خـصـائص الـ DisabledImages والـ HotImages).

- **Indent:** زد هذه القيـمـة لعـرض مـزيد من المسـافـة عند الحـافـة اليـسـرى للـ ToolBar قبل الزر الأول.

- **ShowCaptions:** حـدد هذه القـيـمـة بـ True لعـرض خـصـائص الـ Caption للزرار في الـ ToolBar.

- **Transparent:** حـدد هذه الخاصيـة بـ True لاظهـار لون أو bitmap للـ object الرئيـسي كـخلفيـة للـ ToolBar. عند إدخـال ToolBar في Coolbar، فإنك غالباً سوف تحددها وخاصية الـ Flat بـ True للحصول على المظهر.

- **Wrappable:** عندما تحدد هذه الخاصيةبـ True، تنتقل ازار الـ ToolBar إلى صـفوف جديدة اذا لم تكن جـميعـهـا مـلائمة داخل حـدود الـ ToolBar. ان الـbutton objects الفردية يجب ان يكون لديها خصائص Wrap ايضـاً محـددة بـ True حتى يعمل هذا التأثير بشكل سليم.

خصائص الـ ToolButton:

فيما يلي ملاحظات حول الخصائص المختارة للـ ToolButton. لإضافة زر الى الـ ToolBar، اضـغط زر الفـأرة الأيمن واخـتـر أمـر الـ New Button من القائمة. لاختيار زر منفرد، اضغط صورته في الـ ToolBar، أو يمكنك اختيار الـ ToolButton بالاسم من نافذة الـ Object Inspector.



• **AllowAllUp:** حدد هذه بـ True للسماح لكل أزرار مجموعة ما أن تكون غير مختارة. عندما تحدد بـ False، يجب أن يكون أحد الأزرار المجموعة مختاراً (ويجب أن يتم اختياره أولاً بتحديد خاصية الـ Down له بـ True). وهذه الخاصية ليس لها تأثير إلا اذا كان الـ Style الخاص بالزر محدد بـ tbsCheck، وخاصية الـ Grouped له محددة بـ True.

• **AutoSize:** عندما تحدد هذه الخاصية بـ True، يقوم الزر تلقائياً بتعديل حجمه ليتلائم فى الـ ToolBar. لغالبية التطبيقات، رغم ذلك، يمكنك أن تترك هذه القيمة محددة بـ False وتحدد خاصية الـ AutoSize للـ ToolBar بـ True. وهذان التحديدان يعطيانك فى الغالب التحكم الذى تريده على أحجام الأزرار.

• **Caption:** قم بتعيين label قصيرة لهذه الخاصية، وحدد حقل الـ ShowCaptions للـ ToolBar بـ True، لعرض تعليقات فى الأزار.

على القرص المدمج: • **Command:** قم بتعيين أمراً من الـ CommandList للتشارك مع الزر فى الـ event handler الخاص بهذا الأمر. وهذا هو الـ event handler الذى ينفذ عندما يضغط المستخدم الزر. أو، يمكنك تعيين event handler للـ OnClick الخاص بالـ ToolButton (كما تفعل بالضبط مع الـ Button objects العادية). ولكن باستخدام الـ CommandList تبسط عملية تشارك الأوامر فيما بين objects متعددة- popup menu، قائمة رئيسية، وزر ToolBar، مثلاً. أنظر برنامج العرض Commands على القرص المدمج المرفق بهذا الكتاب فى دليل الـ Source\Commands، وكذلك شرح الـ CommandList component فى الباب الخامس.

• **Down:** تكون هذه الخاصية True اذا كان الزر فى حالته السفلية (مضغوطاً). عند تجميع أزرار متعددة لها Styles محددة بـ tbsCheck وخصائص AllowAllUp محددة بـ True، فعلى الأقل يجب أن يكون زراً واحداً فى المجموعة له خاصية Down محددة بـ True. يمكنك أن تفعل هذا باستخدام الـ Object Inspector، أو بتعيين True لخاصية الـ Down للزر من خلال جملة تكتب فى برنامج. والمكان المناسب لعمل هذا هو OnCreate event handler للـ form.

● **Enabled:** حـدد هذه الخـاصـيـة بـ True لعـرض الزر فى مظهـره العـادى، وكذلك لتشغيل الـ OnClick events له. حددها بـ False لعرض الزر مستخدماً ايقـونتـه اذا كـان قـد تم تعيين واحـدة له. (انظر خـصـائص الـ DisabledImages والـImages للـ ToolBar).

● **Grouped:** عين True لهذه الخاصية لإنشاء مجموعات من الازرار تعمل بالتـفـاعل مع بضـعـهـا البـعض. يجب ان يكون للازرار Styles مـحـددة بـ tbsCheck. حدد الـ AllowAllUp بـ True اذا كانت جميع الازرار مسموح له بأن تبقى غير مختارة؛ والا، فعلى الأقل يجب ان يكون زراً واحداً من المجموعة مختاراً فى كل الاوقات (أى، يجب ان تكون خاصية الـ Down له محددةبـ True).

● **Image Index:** تشير هذه القيمة الى أى من الايقونات يجب عرضها من الـImageList المعين لخاصيـة الـImages للـ ToolBar (وكـذلك خـاصـتين الـ DisabledImages والـ HotImages له اذا كـانتا مسـتـخـدمين). يكون فهرس الصورة الأولى دائماً صفر، والثانية ١، والثالثة ٢ والى آخره. حدد هذه الخاصية بـ 1- اذا لم تكن تريد عرض صورة ايقونة على الزر.

● **Indeterminate:** عندما تحـدد هذه الخـاصـيـة بـ True، يتم عـرض الزر باستخدام لوناً رمادياً، ولكن اذا كانت الـ Enabled ايضاً محددة بـ True، يمكن ان يبقى الزر مختاراً. ان هدف هذا الحقل هو توفير دليل بصرى للمستخدمين ان الزر يمكن ان يكون مختاراً، ولكن قد يكون من غير الصواب ان تفعل هذا بسبب الحالة الراهنة للبرنامج - التحول الى عملية اخرى، مثلاً، اثناء عملية طباعة طويلة. يمكنك ايضاً فحص هذا المجال فى OnClick event handler. للزر، واذا كانت الخاصية True، قم بعرض رسالة تأكيد أو تحذير ان المستخدم قد إختار أمراً غير مناسب.

● **Style:** استخدام هذه الخاصية للاختيار من بين ازرار مختلفة الانواع. ويساوى الـ Style الخاص بالزر العادى tbsButton، وهو التحديد الافتراضى حسب النظام. حدد كل الـ Styles الخاصة بالازرار المجموعة بـ tbsChecked- هذا لا يعرض علامة صح، ولكنه ينشئ زر لاصق يبقى الى اسفل عندما يتم ضغطه

حتى يضغط مرة ثانية . يمكنك ايضاً استخدام هذا الاسلوب لإنشاء ازرار خيار مستقلة- لا يجب ان يتم تجميعها (انظر ايضاً الفقرة التالية) .

- **Wrap**: تحدد هذه الخاصية بـ True لتمكن هذا لزر من الإنتقال إلى على صف جديد اذا لم تكن لتناسب حدود الـ ToolBar . يجب ايضاً ان تكون خاصية الـ Wrappable للـ ToolBar محددة بـ True حتى يعمل هذا التأثير بشكل سليم .

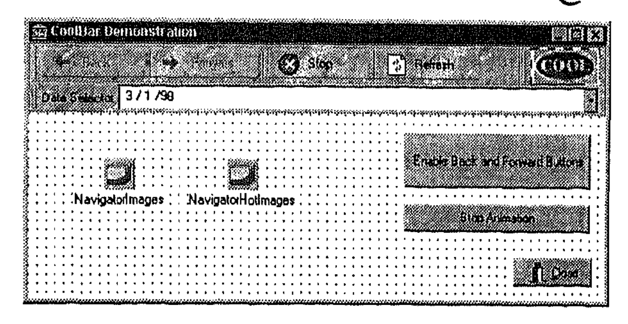
Tip **فكرة:** لإنشاء فاصل بصرى بين الازرار الأخرى، حدد الـ Style بـ tbsDivider أو tbsSeparator (كلاهما نفس الشئ). هذا النوع من الزر هو للعرض الخالص ولا يمكن ضغطه. لعرض سهم مشير الى اسفل على اليمين داخل الزر، حدد كل الـ Styles بـ .tbsDropDown . هذا لا يؤدى الى إنشاء قائمة للزر- لفعل هذا، أضف PopupMenu من Standard palette على الـ form إضغط الـ object مرتين لإنشاء اوامر قائمة، وعين PopupMenu object لخاصية الـ DropDownMenu للزر. ان ضغط هذا النوع من الزر فى وقت التشغيل يؤدى الى فتح القائمة التى يمكن للمستخدمين ان يختاروا منها الأوامر.

الصوروالصور الهامة:

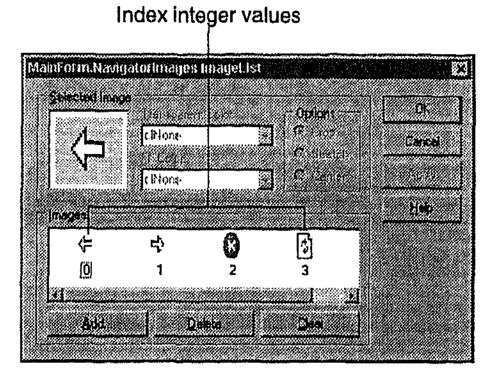
إن الأزرار الموجودة فى الـ ToolBar الخاص بالـ Coolbar يجب أن تعرض جرافيك أيقونى يشير الى ما يفعله الزر. وهذا يتطلب استخدام component آخر من الـ Win32 palette، ألا وهو الـ ImageList. أدخل واحداً أو أكثر من الـ objects على الـ from، ثم استخدم الـ objects للتحكم فى واحدة أو أكثر من الصور الأيقونية للعرض على أزرار الـ ToolBar. (يمكنك استخدام الـ ImageList لأغراض أخرى كذلك عندما تحتاج وعاء تخزين لحمل الأيقونات).

يوضح شكل (٧-١٢) إثنين من الـ ImageList فى نافذة الـ CoolDemo داخل Delphi. و ImageList- ويسميان . NavigatorImages و NavigatorHotImages- يظهران فى وقت التصميم كأيقونات، ولكنهما غير مرئيان عندما يبدأ البرنامج فى التشغيل. لكى أجعل الأيقونات أكثر رؤية هنا، فقد حددت خلفية الـ form باللون الأبيض؛ على شاشتك ستكون الخلفية ملونة بشكل

طبيعى إعتماداً على تحديدات عرض الـ Windows mpct الخاصة بك . لإضافة ، وحذف ، وإعادة ترتيب أيقونات الـ ImageList ، اضغط مرتين ImageList object . هذا يفتح نافذة محرر الـ ImageList الموضحة فى شكل (٧-١٣) .

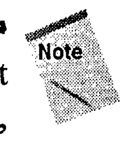


شكل (٧-١٢): الـ form فى وقت التصميم لتطبيق الـ CoolDemo توضح إثنين من الـ ImageList objects كأيقونات والتى تصبح غير مرئية عندما يبدأ البرنامج فى التشغيل



شكل (٧-١٣): افتح نافذة محرر الـ ImageList بالضغط مرتين على الـ ImageList component object

استخدم زر الـ Add فى محرر الـ ImageList لفتح أى أيقونة أو ملف bitmap آخر ، وأدخل الصورة فى القائمة . لتغيير ترتيب الصور الموجودة فى القائمة ، اضغطها واسحبها للمواضع التى تريدها . اختر أى صورة واضغط Delete لحذفها من القائمة . اضغط زر الـ Clear لمسح كل الصور .



ملحوظة: يستطيع محرر الـ ImageList أن يعدل ألوان الـ Transparent والـ Fill color للـ bitmaps أيضاً أن crop أو stretch صورة أو يجعلها فى المركز . اضغط الأزرار المناسبة فى الـ editor لأداء

هذه الأعمال . ولأن الأيقونات ثابتة من ناحية الحجم واللون ، فهذه الخيارات تعتبر مبطلة بالنسبة للأيقونات .

راجع شكل (٧-١٣) . لاحظ أن كل أيقونة قد تم تعيين قيمة صحيحة مفهرسة لها ، بدءاً من الصفر للصورة الأولى . استخدم قيم الفهرس هذه لتعيين خاصية لأزرار الـ ToolBar . وهناك خطوتان لأداء هذا :

١ - اختر الـ ToolBar وعين ImageList object لخاصية الـ Images . هذا يربط الـ ToolBar بمجموعة من الصور - أيقونات الزر فى هذا المثال .

٢ - اختر كل زر منفرد فى الـ ToolBar ، وعين لخاصية الـ ImageIndex الخاصة بالزر قيمة الفهرس للصورة التى تريدها .

هذا هو الحد الأدنى من الخطوات المطلوبة لعرض صور أيقونة على أزرار الـ ToolBar . ولكن قد تريد أن تستخدم طريقتين أخريين لعرض صور الأيقونة . قد تفعل هذا لتوفر صور مختلفة عندما تكون الأزرار مبطلة ، وعندما يحرك المستخدم مؤشر الفأرة فوق الزر . إن استخدام صور مختلفة (أو نفس الصور لكن بألوان مختلفة) .

يستخدم الـ CoolDemo واحداً من هذه الأنماط لتلوين أزرار عندما يشير مؤشر الفأرة إليها . لترى التأثير ، قم بتشغيل البرنامج مرة أخرى وحرك الفأرة فوق الأزرار الأربعة . (اضغط الزر الخاو الكبير لتشغيل زر الـ ToolBar الأوليين) . كما ترى ، يبدو كل زر مضيئاً عندما تمرر مؤشرة الفأرة فوق صورته . يدرك غالبية المستخدمين هذا الدليل البصرى كمشير الى أن الزر "جاهز" للضغط .

لإنشاء هذا التأثير فى الـ CoolDemo ، فقد أنشأت ImageList آخر يسمى NavigatorHotImages (فى الحقيقة هذا هو نفس الـ object الموجود فى برنامج عرض متصفح الـ Web فى Delphi) . والـ ImageList هذا يحمل الأيقونات المتطابقة كالقائمة الأخرى ، ولكن الصور قد تم تلوينها بالألوان الزاهية . إن تعيين الـ ImageList الخاصية الـ HotImages بالـ ToolBar يسبب استخدام هذه الأيقونات عندما تتحرك مؤشرة الفأرة فوق زر فى وقت التشغيل .

والطريقة الثالثة لاستخدام الـ ImageList فى الـ ToolBar هى بتعيين مجموعة ثالثة من الأيقونات لخاصية الـ DisabledImages . إننى لا أفعل هذا فى

الـ CoolDemo لأن الأيقـونات المبطلة حسب النظام الإفتـراضى- المأخـوذة من الـ ImageList ومعينة لخاصية الـ Images ومعتمة تلقائياً- تعتبر كافية فى الغالب. ولكن، يمكنك تعيين DisabledImages اذا أردت استـخدام صور معـينة لأزرار ذات خاصـية Enabled مـحددة بـ False . يمكنك تحريك أيقـونة- لإغلاق باب، مثلاً أو عرض نوع آخر من الصور- عندما يكون الزر مبطل.

Tip

فكرة: لاستخدام الايقونات الموضحة فى تطبيق الـCoolDemo لهذا الباب، فقط انسخ (اختر واضغطCtrl+C) الـ NavigatorImages والـ NavigatorHotImages واحداً فى كل مـرة فى نافذة الـ form الخاص تطبيقك (اضغط Ctrl+V).

الـanimations Coolbar:

من ابسط الاشياء إضافة ايقونة متحركة للـ Coolbar، والتى تشير الى ان البـرنامج يعـمل، وكـذلك إضـافـة flashy graphics للعـرض. يعـرض تطبيق الـCoolDemo قرص "Cool" دائرة، والذى اقترضته من برنامج عرض متصفح Web الخاص بـ Delphi. والصـورة المتـحركـة هى ملف واجـهـة تطبيق قـياسى (avi.)، تم تخزينه فى ملف Source\Data\Cool.avi (وايضاً متوفر مع Delphi فى دليل Demo\CoolStuff)

من السـهل إضافة صورة متحركة للـ Coolbar. ببساطة اسقط Animate component من الـ Win32 palette على الـ Coolbar. تأكد من انك اسقطت الـ Animate على الـ Coolbar، وليس الـ ToolBar. فهـذا يجعل من السـهل وضع الـ Animate فى المكان الذى تريده. (ولـكن يمـكن ان يحـــــمل الـ ToolBar الـAnimate).

ان إسـقـاط الـ Animate على الـ Coolbar ينشئ Band جـديد للصــور المتحركة. فى الغالب ستريد تحريك هذا الـ band الى موضع جديد- مثلاً، يعرض الـ CoolDemo ايقونتة المتحركة الى اليمين من الـ ToolBar الخاص به. لتحريك Band الصور المتحركة:

١- اختر Animate object وحدد خاصية الـ AutoSize له بـ False.

٢- ثم، مازلنا داخل Delphi، اسحب القضيب الرأسى للـ band الى اقصى اليسار، وإسقط الـ Band الى اليمين من الـ ToolBar، أو فى أى مكان آخر تريده. قد تجرب عدة محاولات حتى تحصل على التأثير الذى تريده- من الافضل حفظ المشروع قبل البداية تشكيل الـ Coolbar bands.

٣- عين إسم ملف الـ avi. لخاصية الـ FileName الـ Animate.

٤- حدد خاصية الـ Active بـ True ليبدأ التحرك، والذى يتكرر تلقائياً.

لبدء إيقاف الصور المتحركة فى وقت التشغيل، عين True أو False لخاصية الـ Active. يفعل الـ CoolDemo هذا فى OnClick event handler باستخدام هذه العبارات:

```
Animate1.Active := not Animate1.Active;
if Animate1.Active
  then Button1.Caption := 'Stop Animation'
    else Button1.Caption := 'Begin Animation';
```

يحدد هذا الـ code خاصية الـ Active للحالة العكسية من الـ True أو الـ False لها، ثم تغير تعليق الزر بالنسبة لها.

Note **ملحوظة:** يضع الـ Windows حداً لحجم صورة AVI التى يمكن ان يعرضها الـ Animate control، وبذلك، تنطبق نفس هذه الحصول على الـ Animate لـ Delphi الذى يتوصل لـ Windows control. على سبيل المثال، يجب ان يكون حجم عرض الصورة اقل من 64k ولا يمكن ضغط الملف. ويمكن ان تنطبق حدود اخرى إعتماداً على التشكيل والطول الكلى لملف الـ AVI. إذن لا تكن مندهشاً اذا كان الـ Animate control الخاص بك غير قادراً على تشغيل كل ملف AVI تجده.

Coolbar control اخرى:

كما ذكرت، يمكنك إضافة أى control للـ Coolbar band. والقاعدة الوحيدة هى الـ component يجب ان يكون من class مشتق من الـ TWinControl من أى صلة. على سبيل المثال، يمكنك إضافة ComboBox للـ

لان الـ TComboBox class Coolbar تشـــتق من الـ TComboBox، والتى بدورها تشق من ال TWinControl . ولكن، لا يمكنك إضافة TShape object لـلـ Coolbar لان class TShape الـ TWinControl تـشـــــــــتق مـن الـTWinControl .

ملحـوظة: يمكن من الناحيـة الفنيـة اسقـاط non windowed object control على Coolbar، ولكن هذا لا يؤدى الى إنشاء band جـديد للـ control . للحـصـول على افـضل نتـائج، ضع حـداً لـلـ controls التى تضيفها للـ Coolbar إلى classes components مشتقه من الـ TWinControl .

لإظهار كيف يمكن ان يحمل الـ Coolbar أى windowed control فقد اسـقطت DateTimePicker من الـ Win32 palette علـى الـ Coolbar بالـ CoolDemo . راجع شكل (٧-٩)، أو قم بتشغيل برنامج الـ CoolDemo، لترى هذاالـ control . لان الـ DateTimePicker يسـتـقـر على Coolbar، يمكنك للمستخدمين إعادة تحديد حجم الـ control وتحريكه الى مكان آخر بضغط وسحب القضيب الرأسى للـ band الى اقصى اليسار .

لعرض الـ "Date selector" label الى اليسـار من الـ DateTimePicker، فقد فتحت الـ Bands Editor (بضغط الزر البيضاوى الواقع بعد خاصية الـ Bands للـCoolbar)، واخـتـرت الـ band الذى يحـمل الـ TDateTimePicker . ثم بعـد ذلك ادخلت الـ label فى خاصية الـ band object's Text وعلى عكس كثير من الـ components الأخـرى ذات الـ labels، يجـب ان تسـتـخـدم خـاصـيـة Text - الـ TCoolBand objects ليس لديها حقل Caption .

ملحـوظة: لعـرض labels فى الـ Coolbar bands، يجـب ان تكـون خاصية الـ ShowText التابعة للـ Coolbar محددة بـ True .

الـ StatusBars :

كـمـا اوضـحت سلفاً فى هذا البـاب، ان إحـدى طرق إنشاء status panel بإضافة Panel object على الـ form. وتعطيك هذه التقنية تنوع واسع من خيارات العـرض، ولكن هناك طريقة ابسط للحصـول على status panels حـسنة المظهر

باستخدام الـ StatusBar من الـ Win32 palette . فى غالبية الحالات، يعتبر هذا الـ component هو كل ما تحتاج إليه لإنشاء اشرطة المعلومات status panels فى نوافذ تطبيقك .



ملحوظة: لتفادى التداخل، إننى ادعو عنصر واجهة التطبيق بـ StatusBar عندما يستخدم الـ StatusBar والـ status panel عندما يستخدم الـPanel الخاص بـ Delphi .

إنشاء الـ StatusBar:

تستقر الـ StatusBar عادة فى اسفل النافذة، وهى تقدم معلومات متنوعة عن حالة البرنامج، سجل StatusBar من الممكن أن ينقسم الى Panels (لاتخلط بينها وبين الـ Panel)، كلاً منها يمكن ان يعرض نصاً أو صور جرافيكية . فى اغلب الحالات، يجب ان تعرض معلومات فى الـ StatusBar والتى يحتاج المستخدم ان يجدها بنظرة واحدة مثل ارقام العمود والخط، مواضع الفأرة، احجام الملف، ورسائل الخطأ . يمكنك ايضاً اختيارياً ان تضيف window gripper الى الركن الايمن السفلى من الـ StatusBar– هذا يجعل عملية إعادة تحديد حجم نافذة ذات حد رفيع اسهل .

على القرص المدمج: يوضح شكل (٧-٤) مثال الـ statusbar من مشروع الـ Status، موجود على القرص المدمج المرفق فى دليل الـ Source\Status . لتشغيل البرنامج، ورؤية الـ source code، اكمل الخطوات التالية :

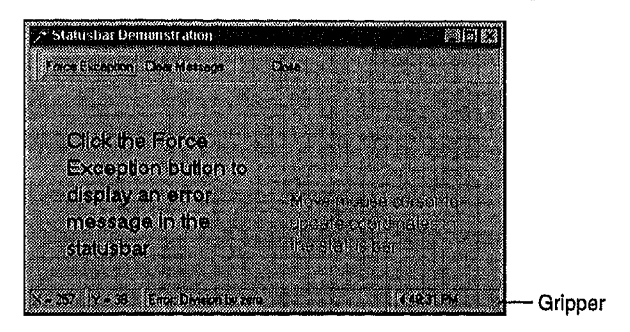
١- افتح ملف مشروع Status.dpr واضغط F9 .

٢- حرك مؤشر الفأرة داخل النافذة لتحديث قيم إحداثيات الـX والـY المعروضة فى اللوحتين الأولتين للـstatusbar .

٣- اضغط زر الـ Force Exception لعرض رسالة خطأ فى منتصف الـ panel .

٤- أختر المستطيل المحيط لإعادة تحديد الحجم بالفأرة واسحبه لإعادة تحديد حجمه تلقائياً، ولكن لوحة الرسالة تغير الطول فقط .

تعمل كثير من هذه السمات فى ظل سيطرة البرنامج، والذى كما سأوضح فيما بعد قائمة البرنامج، يعرض تقنيات عامة عديدة فى برمجة الـ StatusBar .



شكل (٧-١٤): ينقسم الـ StatusBarpanel الى لوحات، كلا منها يمكن ان يعرض نصاً أو صور جرافيكية. لاحظ المستطيل لإعادة تحديد حجم النافذة فى الركن الايمن السفلى من StatusBar

لإنشاء StatusBarإسقط StatusBar من الـ Win32 palette على form .

فى العادى، يقفز object الى اسفل النافذة، حيث يتم عرض غالبية StatusBar ولكن، كما هو الحال مع الـ components الاخرى، يمكنك تغيير خاصية الـ Align لوضعStatusBar على أى حافة من حواف النافذة (alLeft ، alRight ، alTop، أو alBottom) ، أو للملئ الـ client area بالنافذة (alClient) .

يمكن ان يكون لكل object من الـ StatusBar object لوحة واحدة أو أكثر، كلا منها تعتبر object من الـ TStatusPanel class . لإنشاء panels، إتبع هذه الخطوات:

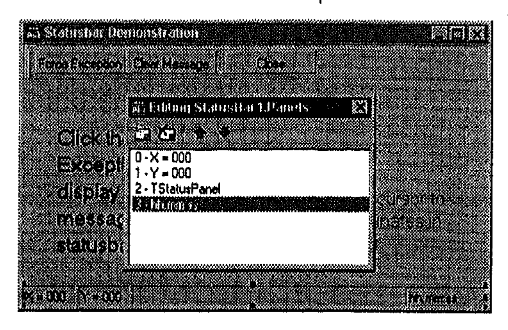
١- اختر StatusBar فى نافذة الـ form .

٢- اضغط الزر البيضاوى الواقع بعد خاصية الـ Panels . هذا يفتح الـ Status Panels Editor [انظر شكل (٧-١٥)].

٣- اضغط ايقونة الـ Add Newلإضافة Panel جديدة.

٤- اضغط Delete Selected لحذف اللوحة التى تم إبرازها. استخدم. ازرار الاسهم المشيرة لأعلى وأسفل لتحريك اللوحة المختارة الى أعلى أو اسفل فى القائمة.

هذا المحرر هو الطريقة الوحيدة للوصول الىpanel objects منفردة في وقت التصميم . باستخدام المحرر، قم بابراز اللوحة التى تريد تعديلها، ثم استخدم الـ Object Inspector لادخال قيم الخاصية .



شكل (٧-١٥): استخدم الـ Status Panels Editor لإنشاء، حذف، وترتيبpanels للـ StatusBar

خصائص الـ StatusBar:

ان الـ StatusBar له خصائص عديدة يمكنك استخدامها لتشكيل الـStatusBar بأكمله . تذكر ان كل panel فى الـ StatusBar التوصل إليها فقط من خلال الـ Status Panels Editor، الذى يفتح عندما تضغط الزر البيضوى الواقع بعد خاصية الـ panel للـ StatusBar . وتتضمن الخصائص المختارة للـ StatusBar مايلى :

- **Canvas:** استخدم هذا الـ object لتلوين النص والصور الجرافيكية فى الـ StatusBar panel . لا يجب عليك ان تفعل هذا الـ text panel النص فقط . استخدم هذا الحقل مع لوحات الرسم، التى سوف اشرحها بالتفصيل قرب نهاية هذا الباب (انظر لوحات الـ StatusBar للرسم) .

- **Panels:** هذه الخاصية هيarray الـ TStatusPanel، كلاً منها يحد مظهر ومحتوى StatusBar panel منفردة . فى وقت التشغيل، اضغط الزر البيضاوى الواقع بعد هذه الخاصية لفتح الـ Status Panels Editor . فى وقته التشغيل، يمكنك التوصل الى الـ StatusPanel objects المنفردة باستخدام هذا array - لتغيير نص panel's text ، مثلاً .

● **SimplePanel**: اذا كنت تريد statusbar احـادى الـ Panel فـقط ، حـدد هذه الخاصية بـ True ودعك من اصطفاف الـ Panel. استخدم الحقل التالى ، SimpleText ، لعرض رسالة فى الـ statusbar . يمكنك ايضاً استخدام هذا الحقل لتحول مؤقتاً statusbar متـعـددة الـ Panels الى واحـد بسيط- لعـرض رسالة طويلة ، مـثـلاً . فـقط ، حـدد الـ SimplePanel بـ True فى عـبـارة برنامج ، وعين نص للـSimpleText . لإعـادة statusbar إلى مظهرة المتعدد الـ Panels ، حدد الـSimplePanel بـ False .

● **SimpleText**: عين أى string للعـرض فى statusbar يكون حـقل الـSimplePanel الخاص به محـدد بـ True لعـرض text فى statusbar مـتـعـدد الـPanels ، استخدم الـ Panels array .

● **SizeGrip**: حدد هذه الخاصية بـTrue لعرض المستطيل المحيط إعادة تحديد حجم نافذة اختيارى فى الركن الايمن السفلى لـstatusbar . هذا يجعل من السهل على المستخدمين إعادة تحديد حجم النوافذ بالضغط والسحب بالفأرة .

خصائص الـ StatusPanel:

تعـتـبـر كل Panel فى الـ StatusBar الـ object من الـ TStatusPanel class. بالرغـم من ان هذا يعـد component ، الا أنه لايظهـر فى الـ VCL palette . ان الطريقة الوحـيـدة لإنشـاء وتعـديل TStatusPanel هو باسـتـخـدام الـ Status Panels Editor (اضغط الزر البيضوى الواقع يعـد خاصيـة الـ Panels لـ StatusBar object's) استخدم المحرر لإضافة ، حذف ، وإعادة ترتيب الـ Panels . اختر Panels فردية ثم استخدم الـ Object Inspector لتعديل الخصائص التالية :

● **Alignment** : تؤثر هذه الخاصية على كيفية عرض الـ text فى الـ panels . حـدد هذه الخـاصـيـة بـ taLeftJustify (وهو التـحـديد الافـتـراضى) لعـرض text flush على الحد الايسر ؛ استخدم الـ taLeftJustify لعرض نص على جانب الحد الايمن ، أو استخدم taCenter لتمركز الـ text داخل عرض الـ panel .

● **Bevel**: للـ panels ثلاث خيـارات عرض . اسـتـخـدم pbLowered (وهو الافتراضى) للحصول على تأثير 3D ، pbRaised للحصـول علىpanel مرتفعة ، أو pbNone للحصول على مظهر مسطح .

- **Style**: حسب المعتاد، حدد هذه الخاصية بـ psText حتى يقوم الـWindows تلقائياً بتحديث محتوى الـ text داخل الـ panel. اذا حددتها بـ psOwnerDraw، فإنك مسؤل عن عرض محتوى الـ panel في الـ OnDrawPanel event handler الخاص بالـ StatusBar، باستخدام خاصية الـ Canvas للـ StatusBar. مؤخراً في هذا الباب، سوف اشرح كيفية استخدام النمط والحدث لعرض صور جرافيكية في الـ StatusBar panels (انظر " لوحات الـ StatusBar للرسم ").

- **Text**: عين text string للعرض في الـ panel. يتم محاذاة النصclasses لخاصية الـ Alignment، ويتم عرضه تلقائياً اذا كانت الـ Style محددة بـ psText. يمكنك استخدام هذه الخاصية لتخزين text panel اذ كانت للـ Style محددة بـpsOwnerDraw، ولكن في هذه الحالة انت مسئول عن عرض النص في الـ panel.

- **Width**: هذه عرض الـ panel بالـ pixels. بالرغم من ان الـ StatusBar يقوم باعادة تحديد الحجم تلقائياً عندما يتغير حجم النافذة، الا إنك قد تريد التحكم في عروض الـ panels الفردية. فمثلاً، قم بتشغيل التطبيق Status واعد تحديد حجم النافذة- لاحظ ان الـ panels الوسطى الكبيرة فقط هي التي يتغير طولها، وتبقى الاخريات بنفس الحجم. وهذا يتطلب بعض البرمجية، كما سأوضح بعد قائمة البرنامج التالية.

التطبيق Status:

On the CD-ROM

على القرص المدمج: ان تطبيق Status، الموجود في دليل ource\Status. على القرص المدمج المرفق بهذا الكتاب، يظهر عناصر رئيسية متعددة في برمجة الـ StatusBar توضح القائمة (٧-٧) الـ source code لبرنامج الـ Status. راجع شكل (٧-١٤) لرؤية عرض البرنامج.

القائمة (٧-٧): Status\Main.pas

```
unit Main;

interface
```

```
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
    Forms, Dialogs, ComCtrls, StdCtrls, Buttons, ExtCtrls,
    ToolWin;

type
    TMainForm = class(TForm)
     Label1: TLabel;
        StatusBar1: TStatusBar;
        Timer1: TTimer;
        Coolbar1: TCoolbar;
        ToolBar1: TToolBar;
        ToolButton1: TToolButton;
        ToolButton2: TToolButton;
        ToolButton3: TToolButton;
        ToolButton4: TToolButton;
        Label2: TLabel;
        procedure FormMouseMove(Sender: TObject;
          Shift: TShiftState; X, Y: Integer);
        procedure FormResize(Sender: TObject);
        procedure Timer1Timer(Sender: TObject);
        procedure ToolButton1Click(Sender: TObject);
        procedure ToolButton2Click(Sender: TObject);
        procedure ToolButton4Click(Sender: TObject);
        procedure Label1MouseMove(Sender: TObject;
          Shift: TShiftState; X, Y: Integer);
          private
     { Private declarations }
      public
        { Public declarations }
    end;
```

```
var
    MainForm: TMainForm;

implementation

{$R *.DFM}

{ Rather than use literal index values, this section defines
  descriptive contants for the statusbar's four panels }
const
  XPanelIndex       = 0;
    YPanelIndex       = 1;
    MessagePanelIndex = 2;
    TimePanelIndex    = 3;

{ Update X and Y coordinate values in the statusbar }
procedure TMainForm.FormMouseMove(Sender: TObject;
  Shift: TShiftState; X, Y: Integer);
begin
    StatusBar1.Panels[XPanelIndex].Text := 'X = ' + IntToStr
   (X);
    StatusBar1.Panels[YPanelIndex].Text := 'Y = ' + IntToStr(Y);
end;

{ Calculate width of middle panel so the others stay the
  same size when the window resizes. }
procedure TMainForm.FormResize(Sender: TObject);
const
  Fudge = 25;  // Allow extra space for width calculation
var
    W: Integer;  // Width of fixed-size panels
begin
    with StatusBar1 do
    begin
```

```
      W := Panels[XPanelIndex].Width +
         Panels[YPanelIndex].Width + Panels[TimePanelIndex].Width;
           Panels[MessagePanelIndex].Width := Width - (W +
      Fudge);
       end;
end;

{ Display the time in the rightmost statusbar panel }
procedure TMainForm.Timer1Timer(Sender: TObject);
begin
  StatusBar1.Panels[TimePanelIndex].Text := TimeToStr(Time);
end;

{ Force an exception to occur and display its message
  in a statusbar panel. The call to ShowMessage is never
     made, but is included to prevent the compiler from
     optimizing out the integer division. }
procedure TMainForm.ToolButton1Click(Sender: TObject);
var
  K, J: Integer;
begin
    K := 100; J := 0;
    try
     K := K div J;  // Force divide by zero exception
        ShowMessage(IntToStr(K) + ':' + IntToStr(J));
      except on E: Exception do
     begin
       MessageBeep(0);
            StatusBar1.Panels[MessagePanelIndex].Text :=
             'Error: ' + E.Message;
        end;
    end;
end;
```

```
{ Clear the text from middle statusbar panel }
procedure TMainForm.ToolButton2Click(Sender: TObject);
begin
  StatusBar1.Panels[MessagePanelIndex].Text := '';
end;

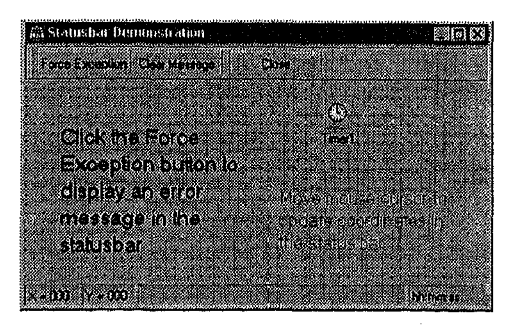
{ End the program }
procedure TMainForm.ToolButton4Click(Sender: TObject);
begin
  Close;
end;

{ Translate mouse coordinates when the cursor moves over
  one of the two large labels, and pass the results on
    to the MainForm's OnMouseMove event handler. }
procedure TMainForm.Label1MouseMove(Sender: TObject;
  Shift: TShiftState; X, Y: Integer);
var
    T: TLabel;  // Refers to Sender as a TLabel object
begin
    T := Sender as TLabel;  // Initialize T
    // Pass coordinates to MainForm's event handler
    MainForm.FormMouseMove(Sender, Shift, T.Left + X,
   T.Top + Y);
end;

end.
```

يعرض برنامج الـ four-panel statusbar فى اسفل النافذة. لإنشاء الـ Panels، فقد قمت بفتح الـ Status Panels Editor (بضغط الزر البيضاوى الواقع بعد خاصية الـPanels للـ StatusBar) واضفت أربع panel objects. لقد حددت أولاً عرض الـPanel الثالثة بـ 250، تاركاً الآخرين محددون بالعرض الافتراضى وهو 50. وكذلك ادخلت نصاً فى كل الـPanels الا الـPanels

الكبرى، حتى تعرض شيئاً فى وقت التصحيح يذكرنى بالغرض من الـPanel. هذا ليس ضـــرورياً رغـم ذلك. يوضح شكل (٧-١٦) مظهـــر الـ form فى وقت التصميم. تكون ايقونة الـ Timer غير مرئية فى وقت التشغيل هذه الـ object توفر الـ Timer event الذى يستـخـدمـه البـرنامج لتـحـديث الوقت فى اقـصى يمين الـPanel.



شكل (٧-١٦): نافذة برنامج الـ Status فى وقت التصميم

لأغراض العرض، تعرض الـ two panels الأولتان فى StatusBar الخاص بالبرنامج إحداثيات X و Y الخاصة بمؤشر الفأرة. قم بتشغيل البرنامج وحرك مؤشر الفأرة داخل نافذة البرنامج- الإحداثيات موضحة فى الـ StatusBar. هناك Two procedures لتحديث هذه المعلومة. الأول OnMouseMove event handler لـ MainForm object's، والذى ينفذ عبارات:

```
StatusBar1.Panels[XPanelIndex].Text := 'X = ' + IntToStr(X);
StatusBar1.Panels[YPanelIndex].Text := 'Y = ' + IntToStr(Y);
```

هذا يوضح كيفية التوصل الى panels منفصل فى الـ StatusBar باستخدام الـ Panels array، يمكن ان تكون قيم الفهرس الموجودة فيما بين الاقواس اعداد صحيحة حرفية (ان فهرس الـ Panel الأولى يكون صفر دائماً)، ولكننى قمت بتحديد ثوابت واصفة مثل XPanelIndex (انظر قطاع الـ const فى منتصف القائمة تقريباً). لانك قد تضيف وتغير، أو تعيد ترتيب الـ panels، اثناء تصميم برنامجك، فمن الصواب ان تستخدم ثوابت واصفة بدلاً من اعداد صحيحة حرفية. اذا قمت بتغيير الـ StatusBar's panels، يمكنك ببساطة تحديث تعينات الثوابت تبعاً لها.

ان إثنان من الـ text objects الكبيـرين بالبـرنامج، الذان يخبـر انك ان تضغط زر الـ Force Exception وان تحرك مؤشرة الفأرة، يتلقيان events تحرك الفأرة الخاصة بهم. لهذا السبب، يعتبر الـ procedure الثانى ضرورياً لتحديث قيم إحداثيات الـ X و الـ Y فى الـ StatusBar عندما تمر بمؤشر الفأرة فوق النص. وهذا يعتبر محيراً بعض الشئ، ويتم تنفيذه فى الـ Label1MouseMove procedure، والملحق هو الآخر بـ label 2 OnMouseMove event و label objects لها نظام الاحدثيات الخاص بها، وهو [0,0] فى الركن الايمن العلوى. لعرض موضع الفأرة بالنسبة للـ client area بالنافذة، من الضرورى ان تضيف موضع كل label [Left, Top] للاحداثيات التى يتقبلها الـ OnMouseMove event للـ labels.

ان تحديث الوقت فى الـ StatusBar panel يعد أمراً بسيطاً. لفعل هذا، قمت بإضافة Timer object على الـ form، وقمت ببرمجة الـ event handler الخاص لها OnTimer لتنفيذ عبارة :

```
StatusBar1.Panels[TimePanelIndex].Text :=
TimeToStr(Time);
```

مرة اخرى، ان استخدام ثابت الـ TimePanelIndex يعتبر أكثر أمناً ووصفاً من قيمة فهرس عدد صحيح حرفية للتوصل الـ Panels array. ان تحديد خاصية الـ Text لهذه الـ panel هو كل ما تحتاجه لعرض الوقت كل ثانية، وهو التردد الافتراضى الـ Timer's event.

لمحاكاة عرض رسالة خطأ- وهى احدى الاستخدامات النموذجية للـ StatusBar panel- يقوم الـ ToolButton1Click بقسمة متغير عدد صحيح على صفر. هذا يجعل جزئية الـ except فى قالب try-except تحدد صفيراً وتعرض حقل الـ Message الخاص بالـ exception باستخدام بالـ code :

```
MessageBeep(0);
  StatusBar1.Panels[MessagePanelIndex].Text :=
        'Error: ' + E.Message;
```

مرة اخرى، لقد استخدمت ثابت واصف للتوصل الى StatusBar panel الصحيحة فى الـ Panels array ان تحديد خاصية الـ Text يعرض رسالة خطأ.

لمسح هذه الرسالة، يعتبر ToolButton event handler آخر null string لنفس هذه الخاصية:

```
StatusBar1.Panels[MessagePanelIndex].Text := '';
```

Note

ملحوظة: عندما تقوم بتشغيل برنامج من داخل Delphi، اذا حدث exception، يعرض مزيل الاخطاء في Delphi, حالة البرنامج الحالى وقد يفتح نافذة CPU. اذا حدث هذا، اختر Tools|Environment Options، اضغط Debugger page tab، وقم بابراز الـ Exceptions الخاصة بـ Delphi في مربع القائمة Exceptions لمنع مقاطعة البرنامج وتشغيله للتعامل مع الـ exception بشكل طبيعي، حدد زر الـ Handled By الى User Program. للعودة الى مقاطعة البرنامج عند تلقى exception، قم بتغيير هذا الزر مرة اخرى الى Debugger.

عندما تعيد تحديد حجم نافذة برنامج Status، يقوم الـ StatusBar تلقائياً باعادة تحديد حجمه ليتلائم مع الحافة السفلية للنافذة. ولكن، هذا التأثير الإفتراضى ليس كافياً للتوصل الى ما أردته- أن أجعل لوحات الوقت والإحداثيات تبقى ثابتة من ناحية الحجم، وأن أجعل الـ panel الكبرى فقط تكبر أو تصغر حسب الحاجة. إن الحصول على هذا التأثير يستلزم بعض البرمجة. لترى هذا، قم بتشغيل Status واحد تحديد حجم النافذة. اختر كيف تتغير الـ StatusBar panel الثالثة فقط فى الطول.

لتوفير مكان للـ code التى تتحكم فى عروض الـ panel يستخدم Status الـ OnResize لنافذة البرنامج. يتم استدعاء هذا الـ event عندما يتم عرض النافذة لأول مرة، وكذلك فى أى وقت يتغير فيه حجم النافذة. لتحديث عروض اللوحة، ينفذ الـ event handler هذا الـ code:

```
with StatusBar1 do
begin
 W := Panels[XPanelIndex].Width +
     Panels[YPanelIndex].Width + Panels TimePanelIndex].Width;
       Panels[MessagePanelIndex].Width := Width - (W + Fudge);
     end;
```

إن متغير W محدد بإجمالى عروض الـ panels الثلاث ذات الحجم الثابت . وهذه القيمة ، بالإضافة الى عامل الـ "fudge factor" ، يتم طرحها من إجمالى عرض الـ StatusBar . والنتيجة تساوى العرض المنشور للـ message panel ، والتى يتم تعينها لخاصية الـ width لهذه الـ panel . وكنتيجة لهذا الـ code ، فإن إعادة تحديد حجم النافذة يتغير فقط طول الـ panel الثالثة . وتبقى panels الوقت والإحداثيات ثابتة الحجم .

:Owner-draw StatusBar panels

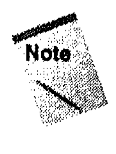
عندما تحتاج مزيداً من التحكم فى كيفية عرض العناصر فى StatusBar panel ، يمكنك الإشارة الى أن panel تحتاج الى تحديث . قد تستغل هذه الفرصة لرسم صور جرافيكية على الـ panels أو لعرض نص باستخدام fonts مختلفة . هذا يسمى StatusBar panel للـ owner-draw (أو ، بتعبير أكثر شمولاً ، owner-draw control) .

فى العادى ، يعتنى الـ Windows بعرض نص فى StatusBar panels . كل ما تحتاج أن تفعله هو أن تعين هذا النص ، مثلاً ، فى event handler . قم بتعيين أى string لخاصية الـ panel object's Text باستخدام عبارة مثل التالية :

```
StatusBar1.Panels[2].Text := 'Insert mode';
```

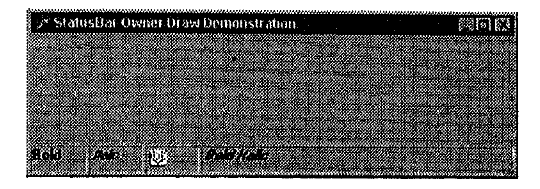
ان تعيين أى string لخاصية الـ Text للـpanel يعرض على الفور هذا النص . اذا احتجت بعد ذلك الى تحديث بالـ panel مثلاً ، عندما يتغير حجم نافذة أو عندما يتم كشفهـا من قبل نافذة اخرى يقوم الـ Windows تلقائيـاً باعادة عرض نص الـ StatusBar panel باستخدام الـ string الذى تم حفظه فى خاصية الـ Text .

لتولى هذه المسئولية ، حدد خاصية الـ Status للـ panel بـ psOwnerDraw باستخدام الـ Status Panels Editor (اضغط الزر البيضاوى الواقع بعد خاصية Panels للـ StatusBar) . يمكنك فعل هذا مع الـ Panels المختارة ، أو معهم جميعاً . مع اختيار هذا النمط ، فمتى يطلب الـ Windows ان تعرض الـ Panels محتوياتهـا ، يتم اطلاق الـ OnDrawPanel event الـ StatusBar object . وهذا الـ procedure ، يمكنك إدخال عبارات البرمجة التى احتاجها لعرض عناصر فى الـ StatusBar panel .



ملحـوظة: ان الـ OnDrawPanel event مـرتبـط الـ StatusBar object . فكل panel فى ذلــك الـ object يجب ان تكون خاصيــة الـ Style لها محــددة بـ psOwnerDraw حتى يتم إطلاق الـ event لهذه الـ panel .

على القرص المدمج: المثال هنا لمعرفة كيفية فعل هذا، وهو مشروع الـ StatusOD (OD تعنى "Owner Draw") الذى يعرض كيفية تغيير انماط الـ fonts لبنود النص فى الـ StatusBar panels ، وايضاً كيفية رسم صورة ايقونة داخل panel . يوضح شكل (٧-١٧) عرض البرنامج . توضح القائمة (٧-٨) الـ source code للبرنامج . توجد ملفات البرنامج على القرص المدمج المرفق بهذا الكتاب فى دليل الـ Source\StatusOD . لتحميل هذا البرنامج فى Delphi ، افتح ملف StatusOD.dpr ، واضغط F9 التشغيل .



شكل (٧-١٧): الـ Owner-draw StatusBar panels تجعل من الممكن تغيير الـ text fonts وعرض صور جرافيكية كما يوضح برنامج StatusOD

القائمة (٧-٨): StatusOD\Main.pas

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
    Forms, Dialogs, Images, ComCtrls;

type
  TMainForm = class(TForm)
```

```
StatusBar1: TStatusBar;
ImageList1: TImageList;
procedure StatusBar1DrawPanel(StatusBar: TStatusBar;
  Panel: TStatusPanel; const Rect: TRect);
private
{ Private declarations }
public
{ Public declarations }
end;
var
  MainForm: TMainForm;
implementation
{$R *.DFM}
procedure TMainForm.StatusBar1DrawPanel(StatusBar: TStatusBar;
  Panel: TStatusPanel; const Rect: TRect);
var
  SavedStyles: TFontStyles;
begin
  with StatusBar, Canvas do
  begin
    SavedStyles := Font.Style;  // Save current font styles
    // Set font style or draw an icon depending
    // on which Panel needs updating.
    case Panel.Index of
    0 : Font.Style := [fsBold];
    1 : Font.Style := [fsItalic];
    2 : ImageList1.Draw(Canvas,
        Rect.Left + 2, Rect.Top + 2, 0);
    3 : Font.Style := [fsBold, fsItalic];
    end;
    // Draw text in all panels except the third,
    // which displays an icon.
    if Panel.Index <> 2 then
```

```
TextRect(Rect, Rect.Left, Rect.Top, Panel.Text);
   Font.Style := SavedStyles;  // Restore saved font styles
  end;
end;

end.
```

يوضـــح الـ StatusBar1DrawPanel Procedure كيفيـــة برمجة الـ owner-draw StatusBar panels. كل panel فى برنامج العرض له خاصية الـ Styleمـحـددة بـ psOwnerDraw. هذا يؤدى الى ان يتلقى OnDrawPanel event handler الخاص بالStatusBar إشارة عندما تحتاج الـ panel الى إعادة الرسم. قد يكون هذا عندما يبدأ البرنامج لأول مرة، و عندما تغير النافذة حجمها، أو عندما تقوم عبارة أخرى فى مكان ما بالبرنامج بتعيين string جديد لخاصية الـ Text للـ panel فى جميع الحالات، يتم استدعاء OnDrawPanel لتحديث محتويات الـ panel. فى هذا المثال، لان الـ StatusBar له اربعة panels ، يتم استدعاء الـ OnDrawPanel اربع مرات: مرة لكل panel، متى يطلب الـ StatusBar تحديثاً.

واحدة من أولى مهـام الـ event handler هى تحديد أى من الـ panel's التى تحتاج إعادة رسم ما عليهـا. ان اسهل طريقة لفعل هذا هى باستخدام خاصية الـ Index للـ Panel parameter. هذا يساوى قيمة الـ index التى تصل الى الـ panel object فى الـ Panels array فى الـ StatusBar. قد يكون افضل اسلوب لفهم هذه العلاقة من الناحية البرمجية - العبارة التالية صحيحة:

StatusBar.Panels[Panel.Index] = Panel;

الـ event handler يتلقى الـ StatusBar parameter. والـــذى يمثل الـ object الذى يحتوى على الـ Panel. إن الـ Panel object يوجد مرجعه فى Panel parameter. وهناك parameter ثالث، وهو Rect، يعطى أبعـاد الـ panel على الشاشة- اذا كانت النافذة تغير حجمها، فإن هذا السجل يحتوى على الحجم الجديد للـ panel.

لرسم نص بأنماط مختلفة، يعين البرنامج مجموعات من ثوابت النمط. لخاصية الـ Font التابعة لـ StatusBar.Canvas. إننى أقفز هنا بعض الشئ، إن

مفهوم الـ Canvas سيتم شرحه فى الباب الثالث عشر، وهو تطوير تطبيقات الجرافيك. تخيل الـ Canvas على أنه سطح تستطيع أن ترسم عليه control فى حالة الـ StatusBar، يقدم الـ Canvas السطح الذى ترسم عليه أو تلون نصاً داخل panel منفردة.

تقوم عبارة with فى الـ procedure بتبسيط بعض الـ code. على سبيل المثال، لتعيين نمط بنط الـ fsBold لخاصية الـ Font.Style للـ Canvas، فإن عبارة الحالة تنفذ العبارة:

```
Font.Style := [fsBold];
```

إن أنماط البنط تعتبر مجموعات، ولذا، فبالرغم من أن هناك قيمة واحدة فقط محددة، فيجب أن تكون بين أقواس. اذا لم يتم استخدام عبارة with، فلن تكتب هذه العبارة:

```
StatusBar.Canvas.Font.Style := [fsBold];
```

كلتا العبارتين متساويتـــــان، ولكن عبارة with تبسط العبارة الأولى باخبار الـ Delphi أن يستخدم خاصية الـ Font فى الـ Canvas الخاص بالـ StatusBar وهذا حسب النظام الإفتراضى. بعد تحديد نمط البنط، يستدعى البرنامج الـ Canvas method التابع للـ StatusBar لعرض نص باستخدام هذا النمط. وهذه العبارة:

```
TextRect(Rect, Rect.Left, Rect.Top, Panel.Text);
```

ترسم قيمة الـ Text الخاص Panel object باستخدام الـ Rect وقد تم تمريره كـ parameter event handler. ان اول ذكر للـ Rect هنا يحدد الحدود لقدرة الانتاج- اذا زاد رسم النص عن حجم الـ panel، فيتم قصه ليتناسب. وتحدد القيمتين الجديدتين موضع الركن الايسر العلوى لبند النص. والـ parameter الاخير بالطبع هو الـ string الذى سوف يرسم.

ان رسم الصور الجرافيكية فى panel يتم بنفس السهولة. لفعل هذا فى البرنامج، تستدعى جملة الـ Draw method فى الـ ImageList object الخاص بالبرنامج:

```
ImageList1.Draw(Canvas,
  Rect.Left + 2, Rect.Top + 2, 0);
```

لا يجب عليك ان تستخدم الـ ImageList object ، ولكنها ابسط طريقة لادخـال الـ bitmaps، وبخـاصـة فى حـالة الايقـونات . يقـوم الـ ImageList component بفعالية بتخزين مثل هذه الصور لـbitmaps فردية فى الذاكرة . قم بتحديد رقم الـ index للصـورة التى تريدها (وهو صـفر فى هذه الحـالة، وقد تم تمريره كـاكبر قيمة متغيرة للـDraw) ، ويسترجع الـ ImageList object هذه الصـورة. لاحظ كيف تحددعبارة Draw الـ Canvas الخاص بـ StatusBar1 .object

أفكار للمستخدم الخبير

● عين قيمة لخاصية الـ HintPause للـ application object لتغيير تأخير الوقت ليظهر مربع hint box . ان القيمة الافتراضية هى ٨٠٠ ملى ثانية . على سبيل المثال، لبرمجة ظهور اسرع للـ hint box، يمكنك تعيين ٢٥٠ (أى ٤/١ ثانيـة) للـApplication.HintPause . قـد تفـعل هذا فى أى event handler (استخدم الـ OnCreate للـ form لتوصيف الـ hints بمجرد بدء التشغيل) .

● ان لون الـ hint box اصفر والنص بها اسود، وهو ما يمثل الـ Post-It® Notes الاصلية . لتغيير هذا اللون، يمكنك تعيين قيمة TColor لخاصية الـ HintColor للتطبيق . فمثلاً، لإنشاء hint box حمراء، عين clRed للـ Application.HintColor . (انظر نوع بيانات الـ TColor فى الـ online help الخاصة بـ Delphi لمعرفة المزيد من المعلومات) تذكر ان المستخدمين باستطاعتهم ان يغيروا لون الـ hint box بتعديل تحديدات نظام الـ Windows .

● عند تصميم status panels مقسمة وتملك أكثر من TPanel component، إختر font للـ panel الرئيسية، ثم حدد خاصيةالـ ParentFont بـTrue لكل panel فرعية . عندئذ، يستخدم كل تقسيم فرعى تحديدات بنط الـ panel الرئيسية . فهذا اسهل من تعيين خصائص البنط لكل panel فرعية .

• يمكنك استخدام TPanel objects كاسطح مرتفعة لعرض نص فى أى مكان من نافذة الـ form- لا يجب عليك قصر الـ TPanels على الاستخدام لـ toolbar و status panel على سبيل المثال، يستخدم الـ Compiling information dialog الخاص بـ Delphi الـ TPanels لعرض المعلومات اثناء الـ Compiling. لرؤية هذه النافذة، إختر امر الـ Options|Environment، اختر باب الـ Preferences page tab لعرض الـ compiler progress وقم بإجراء عملية الـ compile لمثال.

• أضف Panel على الـ form وحدد خاصية الـ Align لها بـ alClient. هذا يجعل الـ Pane تملأ نافذة الـ form تماماً. اختر قيم للـ BevelInner والـBevelOuter، وقم بتحديد أكثر من قيمة للـ BevelWidth والـ BorderWidth (١٠ و ٢٥، مثلاً). يمكنك الحصول خلفيات نافذة ممتعة باستخدام الـ Panelكخلفية نافذة. (اذا اضفت الـ components، اختر الـ Panel واختر Edit|Send to Back).

• بدلاً من استخدام الـ glyph bitmap، يمكنك استخدام خاصية Font للـ SpeedButton لعرض رموز. حدد الـ Font الخاص بالزر ببنط الـTrueType مثل الـ WingDings. لايزال الـ Caption يعرض رموز نص ASCII. هذه التقنية تعمل مع أى عنصر نص له خاصية Font.

• عند استخدام الـ Format function، فالخطأ الشائع هو ان تنسى إدخال الـ arguments فى الاقواس. تذكر ان Args parameter الخاص بالـ Format يعد array- بعبارة اخرى، قائمة parameter متغيرة الاطوال. على سبيل المثال، يمرر التعبير [V1,V2,V3] للـ Args المتغيرات V1، V2، و V3.

• ان كلاً من TCoolbarclasses و الـ TToolBar classes تنحدر، من من TToolWindow classes، والتى تحدد السمات لنافذة ذات حدود و client area مخصصة. هذا يجعل الـ TCoolbar و الـ TToolBar اقارب (يتشاركون فى نفس الـ classes الأم). لمعرفة المزيد عن الـ TCoolbar components و الـ TToolBar، قد تريد ان تتعرف على online والتوثيقات المطبوعة للـ TToolWindow class.

● الـ Coolbar عـبـارة عن windowed control، وقـد يتم إدخـاله فى Coolbar آخر لإنشاء multilevel concoction. قد تستخدم هذه التقنية لإنشاء Coolbars ذات bands مرتبة رأسيا واخرى مرتبة افقيا- فى الواقع، هذه هى الطريقة الوحيدة للحصول على هذا التأثير. ولكن، تأكد من انك فعلاً تحتاج الى هذه التعقيدات الخاصة بالـ multilevel Coolbar قبل ان تبذل مجهوداً فى تنفيذ الفكرة.

المشروعات التى يمكنك تجربتها

(٧-١): قم بإنشاء toolbar يمكن للمستخدمين تحريكه الى الحد العلوى للنافذة، أو الحد السفلى، أو الايمن، أو الايسر بالإضافة الى تعيين قيمة جديدة لخاصية الـ Align للـ Panel، يجب عليك ان تعدل خصائص الـ Left والـ Right للـ SpeedButton الى مواضع جديدة قريبة داخل حدود الـ Panel object بـ toolbar.

(٧-٢): قم بانشاء الـ status panel توضح التاريخ والوقت. (ملحوظة: راجع الـ source code لتطبيق الـ Toolbar2). قم باختبار تطبيقك بتشغيله طوال الليل للتأكد من انه يغير التاريخ فى منتصف الليل.

(٧-٣): قم بتصميم floating toolbar لاختيار اللون، وذلك باستخدام تطبيق الـToolbar3 كمرشد لك.

(٧-٤): قم بتجربة الضغط والسحب للـ objects مثل الـ Buttons والـ SpeedButtons فى وقت التشغيل. استخدم برمجة شبيهه بتلك الموجودة فى Toolbar3 للاستجابة للـ events والـ OnMouseDown، والـ OnMouseMove، والـ OnMouseUp.

(٧-٥): اكتب برنامج اختبار مع الـ status panel التى تعرض رسالة خطأ. استخدم الـ Format function لإنشاء integer error

code . اضف ازرار واوامر قائمة للبرنامج لمحاكاة انواع مختلفة من الاخفاء .

(٧-٦) : ادخل التاريخ والوقت فى الـ status panel .

(٧-٧) : قم بتصميم form template مع الـ status panel من التطبيق CapsLock [راجع شكل (٧-٨) و القائمة (٧-٥)] .

(٧-٨) : قم بتحويل برنامج الـ Tabs الموجود على القرص المدمج فى دليل الـ Source\Tabs ليستخدم الـ StatusBar objects والـ ToolBar الخاصة بالـ Win32 .

ملخص:

● استخدم الـ Panel component لإنشاء toolbars و status panels . تحتوى الـ toolbars على SpeedButton objects . تعرض Status panels نصاً .

● قم بتجميع الـ StatusBar objects فى مجموعات بتحديد خصائص الـ GroupIndex لها بنفس القيمة العددية الصحيحة الموجبة غير الصفرية . تعمل الـ SpeedButton بعد ذلك مثل الـ RadioButtons- يمكن ان يكون زراً واحداً فقط فى المجموعة الى اسفل . حدد الـ AllowAllUp بـ True لتسمح لجميع الازرار بالمجموعة ان تبقى فى حالة إغلاق .

● قم بتصميم floating toolbars بإنشاء الـ OnMouseDown ، OnMouseMove ، و OnMouseUp . قم باستدعاء الـ SetCapture function الخاصة بالـ Windows لتبدأ عملية ضغط وسحب . دائماً إجعل كل استدعاء للـ SetCapture ملحق باستدعاء للـ ReleaseCapture . انظر الباب الثانى عشر لمزيد من المعلومات عن امكانيات نافذة الوصل الجديدة والتى يمكنك استخدامها ايضاً لإنشاء floating toolbars .

● هناك تشابه بين الـ status panel و toolbars ولكنها غالباً ما تعرض نصاً بدلاً من الازرار . قم بتقسيم status panel بادخال Panel objects فى الـ Panel الرئيسية . قم بعرض نص فى التقسيمات الفرعية للـ Panel بتعيين strings

لخاصيــــــة الـ Caption بالتبادل ، ادخل الـ Label objects فى الـ Panel فرعية وعين نصاً للـ Caption الخاصة بالـ Label- يمكنك أيضاً عرض نص معتم فى الـ status panels بتحديد خاصية الـ Enabled للـ Label objects بـ False .

- استخدم الـ Format function بـ Delphi لإنشاء formatted strings . تشبه هذه الـ function الـ .sprintf() بالـ C والـ C++ . تعد الـ Format مفيدة فى إنشاء strings للعرض فى status panels ، ولكنها و function نافعة ايضاً لكثير من الاغراض .

- تعـرض الـ status panels الكلاسكيـة مـفـاتيح الـ Caps Lock ، الـ Num Lock، و الـScroll Lock ، والـ Ins ، كـمـا يوضح برنامج الـ Caps Lock الخـاص بهـذا البـاب ، استـدع الـ Windows GetKeyState function للحصول على المواصفات المرتبطة بالمفاتيح .

- يمكن لـ Coolbar من لوحة الـ Win32 ان يحمل band من windowed control مثل الـ Win32 ToolBars وقائمة اللائحة . يستطيع المستخدمون إعادة ترتيب الـ Coolbar bands فى وقت التشغيل .

- يقـدم الـ StatusBar من لوحـة الـ Win32 طريقـة بسـيطة لإنشـاء الـ statusbar متعدد الـ bands فى النافذة .

فى الباب التالى، ستعرف كيفية استخدام الـ Delphis list component مثل الـ ListBox والـ ComboBox . سوف تفحص أيضــاً الـ list classes مثل الـ TList، الـ TStrings ، و الـ TStringList .

الباب الثامن
تكوين الـ Lists

محتويات هذا الباب:

- Components
- List components
- String and other lists

ان كان هناك هيكل بيانات واحد يستخدمه المبرمج الآن أو فيما بعد، فكل مبرمج سوف يستخدم القائمة. فليس من المثير للدهشة ان واحداً من اهم الـ Windows controls هو list box. ومع هذا الـ control، والـ control المرتبط به وهو الـ combo box، والذى يضيف text-entry area لـ list box، يمكنك إنشاء string list لأية اغراض اخرى. يمكنك ترتيب بيانات الـ list box ويمكنك الاختيار من بين انواع انماط مختلفة للـ list box و الـ combo box- فمثلاً، ان تسمح بالاختيار المتعدد، .

وكذلك يقدم Delphi عدة classes لصنع lists متعددة متعلقة ببعضها، مثل الـ TList، الـ TStrings ، الـ .TStringList . هذه هى كل الاسلحة النافعة فى معركة إنشاء واجهة تطبيق معقدة للمستخدم، فى معركة إنشاء واجهة تطبيق معقدة. للمستخدم، وكما ستعرف فى هذا الباب، ان Delphi يقدم مجموعة غنية من الادوات للحفاظ على أى نوع من الـ list .

:Components

فيما يلى قائمة الـ Delphi's components لصنع الـ list :

- **ComboBox**: يجمع الـ Windows control المعيـــــارى هــــــذا بين الـ ListBox و الـ Edit object . وإعـتـمـاداً على نمط الـ ComboBox object، يتسطيع المستـخـدمـون الاختيـار من بين المداخل المذكـورة فى الـ list، والتى تظهـر اخـتـيـاريـاً فى drop-down، أو يمكنهـم إدخـال بيـانات جـديدة فى الـ Edit control . الـ Palette : Standard .

- **ListBox**: استـخـدم الـ Windows control المعيـارى هذا لإنشـاء strings lists والتى يمكن للمستخدمين ان يختاروا منها باستخدام اوامر الفأرة ولوحة المفاتيح . كـمـا سيـوضـح هذا البـاب، تقـدم الـ ListBox فى TStrings array يسـمى Items والذى يوفر وصولاً اسهل لبيانات الـ ListBox. الـPalette : Standard .

- **StringGrid**: بالرغـم من اسـمـهـا، فـإن هذا الـ component قـادراً على تخزين بيانات أو bitmap أو غيرها. وهـذا يجعـل الـ StringGrid نافعـة بشكل خاص فى إنشـاء lists للـ objects ذات الاسـمـاء، كـمـا هو موضـح فى تطبيق هذا البـاب الـ GlyphLst، والذى يعـرض اسـمـاء ملفـات bitmap لكل صـورة ايقـونة ملونة glyph يقدمها Delphi . الـPalette : Additional .

List Components :

ان الـ Delphi components المعـــيـــارين، وهـمـــا الـ ListBox والـ ComboBox، يمكن ان يتوليـان أمر متطلبات اختيـار الـ list الخاصة ببرنامجك كله. كـمـا ستعرف فى هـذا الفصل، ان Delphi يدعم الـ controls ليوفـر طريقة للوصـول إلى البـيانات الواردة فى الـ list. على سبـيل المثـال، بعـبـارات بسيطة، يمكنك نقل الـ ListBox strings والـComboBox من والى ملفـــات النص . والتطبيق الخاص بهذا الفصل- وهو الـ ToDo List utility- يوضـح هذه التقنية، وكذلك يوضح كـيفية نقل strings بين اثنين من الـ ListBox باسـتـخدام واحدة من dialog-box templates الخاص بـ Delphi. يمكنك ان تجـد تطبيق الـ ToDo فى دليل الـ Source\ToDo على القرص المدمج.

الـListBoxes :

يقدم الـListBox عـدداً من الخصائص الهامة التى يمكنك من الاختيـار من بين انماط مـخـتلفـة. على سبـيل المثـال، عـادة مـا تكـون الـListBox لديهـا حـد

خارجى، ولكن يمكنك تحديد الـBorderStyle بـ bsNone لتحصل على قائمة بلا حدود. (هذا التشكيل يبدو جيداً مع الـListBox المدخلة فى Bevel). عين قيمة لخاصيـة الـColumns لعـرض عناصـر فى list فى اكثر من عـمـود. حسب النظام الافتراضى ، هذه القيمة محددة بصفر- حددها باثنين أو اكثر لإنشاء قوائم ذات اعمدة.

هناك طرق مـخـتلفـة عـديدة لإدخـال بيـانات فى الـListBox . فى وقت التصميم، اضغط الزر البيضاوى لخاصية الـ Items لفتح الـ String list editor الخاص بـDelphi وادخل بنود قـائمـتك. أو، يمكنك نسخ نص من ملف آخـر فى الـeditor هذا الـ method يخـزن الـstrings المذكـورة فى قـائمـة فى ملف exe. ويدخل تلقائياً الـ strings فى الـListBox .

لإضـافـة strings فى الـListBox فى وقت التشـغيل، يمكن ان يسـتـدعى البرنامج methods لخاصية الـItems من الـ TStrings class كل ListBox يقدم Items object للتـمكن من توصل اسـهل الى بيـانات الـlist . على سبيل المثال، لحث المستخدمين على إضافةstrings جديدة فى الـListBox ، يمكنك استخدام البرمجة التالية مثل :

```
var
    S: String;
begin
    S := InputBox('Test Program', 'Enter a string', '');
    if Length(S) > 0 then
     ListBox1.Items.Add(S);
end;
```

لاحظ انك تسـتـدعى الـ Add method للـ Items object الذى يملكه الـ ListBox1. ان الـ ListBox نفسـه ليس له Add method- ، هذا الـ method ينتمى للـ Items. ولكن، يمكنك تبسيط الـ code السابق باستخدام عبارة with مثل هذه :

```
with ListBox1.Items do
  Add(S);
```

ان عبارة with تخبرالـ Delphi ان يستخدم method والتعريفات الاخرى فى الـItems حسب النظام الافتراضى للحصول على strings out للـ ListBox، استخدم الـItems كـ strings . على سبيل المثال، حدد متغير كـ strings مثل :

```
var
  S: String;
```

بعـد ذلك، عـين أى ListBox string للـ S بتـحـديد الـ integer index للـ Item مع العبارة، وعملية الوصول إلى الـItems فى الـ array of Strings :

```
S := ListBox1.Items[0];
```

ان المظاهر خداعة فى هذه الحالة لان الـ Items ليس array فى الحقيقة- إنها خـاصيـة TListBox class التى تستـخـدمهـا بنفس طريقـة الـ Pascal arrays . لتخزين بياناتها الـ Items تملك خاصية Strings array مستخدماً فى عبارات تحدد قيم الـ index فيما بين قوسين . والعبارة التالية، على سبيل المثال، مساوية تماماً للعبارة السابقة :

```
S := ListBox1.Items.Strings[0];
```

لان الـ Strings array هو الافتـراضى للـ Items، فإن الأسـمـاء لا تحتـاج الى تحديد . وفعل هذا لا يضر، لذلك استخدم أى نوع من العبارة التى تفضلها . أى كان الـ method الذى تستخدمه لفهرسة الـ ListBox's strings، إقصر الفهرس دائماً على المدى من (0) الى Items.Count - 1 . اذا لم تفعل، الـ Delphi يحدث exception

استخدم for loop، على سبيل المثال، للتوصل لكل الـ ListBox's strings :

```
for I := 0 to ListBox1.Items.Count - 1 do
begin
  S := ListBox1.Items[I];
   ShowMessage(S);  { Or do something else with S }
end;
```

بالنسـبـة للـ lists الطويلة، قـد تريد ان توفـر بيـانات من ملف نص يمكن للمستخدمين اعداده وتحريره مع الـ Windows Notepad . لتحميل ملف النص فى الـ ListBox، استـدع الـ LoadFromFile method للـ Items كـمـا فى هذا المثال :

```
ListBox1.Items.LoadFromFile('C:\data\file.txt');
```

استدع الـ AddStrings لإضافة strings متعددة من احد الـ ListBox (أو أى component آخر ذا خاصية الـ TStrings) الى آخر. على سبيل المثال، هذه العبارة تلحق الـ stringsا من الـ ListBox2 الى أية strings موجودة بالفعل فى الـ ListBox1 :

```
ListBox1.Items.AddStrings(ListBox2.Items);
```

لتبادل بندى الـ list، استدع Exchange، كما فى هذه الـ code، والذى يبذل الـstrings الأول و الاخير فى الـ ListBox1 :

```
with ListBox1.Items do
  if Count >= 2 then Exchange(0, Count - 1);
```

لاحظ، مرة اخرى هنا، ان مدى الـ strings هو من قيم الـ index من (0) الى Items.Count - 1 لحذف strings من الـListBox، استدع الـ Delete method، والذى يتطلب integer index items. على سبيل المثال، تحذف هذه العبارة البند الرابع من الـ ListBox1 (ان فهرس البند الأول هو صفر، لذا فإن الرابع يتم تعريفه بقيمة فهرس تساوى٣) :

```
ListBox1.Items.Delete(3);
```

بدلاً من تخصيص فهرس حرفى، أو فى الحالات التى لا تعرف فيها فهرس البند، لكى تجد قيمة الفهرس، استدع IndexOf، والذى يدخل 1- اذا كان البند المحدد لا يوجد :

```
Index := ListBox1.Items.IndexOf('Item to find');
if Index >= 0 then
begin
  { ... ok to use Index }
end;
```

استدع الـ Clear method لحذف كل البنود من القائمة :

```
ListBox1.Items.Clear;
```

Scrolling in listBoxes:

تأتى الـ ListBoxes بمفاجأة- اذا كان هناك سطور للعرض اكثر من المتاح داخل مساحة الـ ListBox، يظهر scroll bar رأسى تلقائياً. عندما يستطيع الـ ListBox عرض جميع البنود، يختفى scroll bar الرأسى تلقائياً.

تؤثر خاصتيان على الـ scroll bar الرأسى فتشير الـItemHeight الى ارتفاع لسطر النص الواحد بالـ pixels. حدد الـ ItemHeight بـ True لضمان ان سطور النص الكامل فقط تظهر فى حدود الـ ListBox. حدد الـ ItemHeight بـ False لتسمح بسطور جزئية فى اسفل الـ ListBox.

عندما تختار الـ font للـ ListBox، أو لـ form تحتوى علىListBox، يقوم Delphi تلقائياً بتعديل الـ ItemHeight الى قيمـــــــة مقـــابلة. للتغلب على هــــــذا التعديل التلقائى، يجب ان تحدد الـ Style بـlbOwnerDrawFixed أو بـ lbOwnerDrawVariable. لمزيد من المعلومـات عن انشــاء owner-draw controls، انظر "استخدام الـ TStrings class" لاحقاً فى هذا الباب.

يمكن ايضاً ان تعرض الـ ListBoxes scroll bar افقى. لتشغيل الـ scroll bar الافقى، استدع الـ Perform method التابعة للـ ListBox، والذى يرسل رسـالة الى الـ Windows ListBox الذى يقـدم له الـ ListBox component واجهة تطبيق مختصة بالـ objects. والرسالة التى يتم ارسالها فى هذه الحالة هى LB_SETHORIZONTALEXTENT. ومع الرسالة، قم بتمرير مدى تحريك افقى بالـ pixels، زائد قيمة متغيرة ثالثة لم تستخدم والتى تكون دائماً صفر. على سبيل المثال، هذه العبارة تضيف scroll bar افقى للـ ListBox، باقصى مدى تحريك وهو ١٠٠٠ pixels:

```
ListBox1.Perform(LB_SETHORIZONTALEXTENT, 1000, 0);
```

بدلاً من تمرير مدى حرفى مثل الـ ١٠٠٠، فإن برنامجاً افضل قد يحدد مدى التحريك الافقى بعرض اطول الـ strings، بالـ pixels. توضح القائمة (٨-١) كيفية تحديد هذه القيمة. يستدعى البرنامج الـ TextWidth فى الـ Canvas الخاص بالـ ListBox (object يساعدك للوصول الى واجهة التطبيق الجرافيكية أو الـ GDI). يدخل الـ TextWidth عـرض الـ string، بالـ pixels، للـ font

الحـــالى للـ Canvas . بعد تنفيذ الـ for loop، يساوى متغير الـ K اقصى عرض الـ string، بالـ pixels، والذى يمررة البرنامج للـ ListBox مستخدما رسالة الـ Windows : LB_SETHORIZONTALEXTENT.

On the CD-ROM

على قرص المدمج: لكى تجعل مدى scroll bar الافقى متماشياً مع بيانات الـ ListBox، قم باداء الخطوات الموضحـــة فى القـــائمة (٨-١) بعد كل إضافة أو حذف، بعد تغيير الـ font الـ ListBox. على القـــــرص المدمج، توجد هذه القائمة فى دليل الـ Source\Lists فى ملف Adjust.pas . استخدم هــــذه القائمــــة لتعديل مدى الـ scroll bar الافقى الخاص بالـ ListBox الى عرض اطول list string ، بالـ pixels.

القائمة (٨-١): Code لتعديل مدى scroll bar افقى للـ ListBox

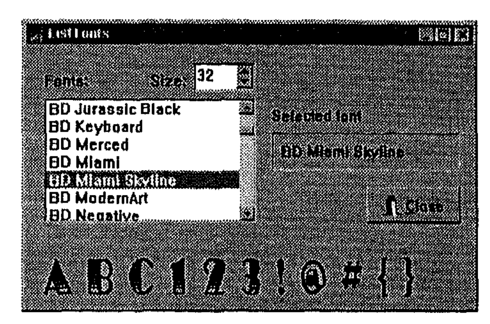
```
Tvar
 I, J, K: Integer;
begin
 with ListBox1 do
    begin
     K := 0;
       for I := 0 to Items.Count _ 1 do
       begin
        J := Canvas.TextWidth(Items[I]);
          if J > K then K := J;
        end;
       Perform(LB_SETHORIZONTALEXTENT, K, 0);
      end;
end;
```

ان الـ scroll bar افقى لا يوصى بها للـ ListBox متعددة الاعمدة. ولكن، قـــد تعدل عرض العمود بارسال رسالة الـ LB_SETCOLUMNWIDTH للـ ListBox باستخدام عبارة هذه:

```
ListBox1.Perform(LB_SETCOLUMNWIDTH, 100, 0);
```

انشاء الـListBox:

ان تطبيق الـ ListFont الخاص بهذا الكتاب على القرص المدمج فى دليل الـ Source\ListFont يوضح كيفية إدخال قائمة باسماء font النظام فى ListBox ، ثم استخدام تلك الاسماء لعرض string فى كل اسلوب font. يوضح شكل (٨-١) عرض البرنامج. قم بتشغيل الـ ListFont واختر أى font لعرض النص بالنمط الافتراضى للـ font. اضغط زر الـ size لاختيار حجم نقطة جديد. تعتبر الـ TrueType Fonts فقط هى التى يمكن قياس كل أحجامها. توضح القائمة (٨-٢) الـ source code للبرنامج.



شكل (٨-١): يحمل الـ ListFont اسماء Font النظام فى الـ ListBox

لإضافة list باسماء البنط فى الـ ListBox، ينفذ برنامج الـ FontList العبارة التالية فى OnCreate الخاص بالـ form:

```
FontListBox.Items := Screen.Fonts;
```

لتصنيف قائمة الـ font أبجدياً، حدد خاصية الـ Sorted للـ ListBox بـ True.

القائمة (٨-٢): Listfont\Main.pas

```
unit Main;

interface

uses
  SysUtils, Windows, Messages, Classes, Graphics,
```

```
  Controls, Forms, Dialogs, StdCtrls, Spin, Buttons, ExtCtrls;

type
 TMainForm = class(TForm)
    FontListBox: TListBox;
      Label1: TLabel;
      SampleLabel: TLabel;
      SpinEdit1: TSpinEdit;
      Label3: TLabel;
      FontNameLabel: TLabel;
      Label2: TLabel;
      Bevel1: TBevel;
      CloseBitBtn: TBitBtn;
      procedure FormCreate(Sender: TObject);
      procedure FontListBoxDblClick(Sender: TObject);
      procedure FontListBoxKeyDown(Sender: TObject;
       var Key: Word; Shift: TShiftState);
      procedure SpinEdit1Change(Sender: TObject);
      procedure CloseBitBtnClick(Sender: TObject);
     private
    { Private declarations }
     public
    { Public declarations }
     end;
  var
   MainForm: TMainForm;
  implementation
  {$R *.DFM}
  procedure TMainForm.FormCreate(Sender: TObject);
  begin
     FontListBox.Items := Screen.Fonts;
  end;
```

```
procedure TMainForm.FontListBoxDblClick(Sender:
TObject);
begin
  with FontListBox do
     if ItemIndex >= 0 then
     begin
       SampleLabel.Font.Name := FontListBox.Items[ItemIndex];
        FontNameLabel.Caption := SampleLabel.Font.Name;
      end;
end;

procedure TMainForm.FontListBoxKeyDown(Sender: TObject;
    var Key: Word; Shift: TShiftState);
begin
  if Key in [VK_RETURN, VK_SPACE] then
       FontListBoxDblClick(Sender);
end;

procedure TMainForm.SpinEdit1Change(Sender: TObject);
begin
     SampleLabel.Font.Size := SpinEdit1.Value;
end;

procedure TMainForm.CloseBitBtnClick(Sender: TObject);
begin
  Close;
end;

end.
```

تستخدم الـ FontList تقنيتين لإختيار اسماء الـ font من الـ FontListBox. فى OnDblClick ، تقوم عبارة if أولاً بفحص ما اذا كانت خاصية الـ ItemIndex للـ ListBox أكبر من أو تساوى صفر. اذا كانت كذلك، فيكون المستخدم قد اختار بنداً من الـ ListBox ولم يضغط مرتين الهواء الساخن.

وتقوم العبارتان المتتاليتان بتعيين اسم الـ font المختار من الـ Items لخاصية الـ Font.Name التابعة للـ SampleLabel . عند تعيين Name جديد لخاصية الـ font يتغير الـ Font للنص المعروض فى اسفل النافذة. وأخيراً، يحدد الـ FontList اسم الـ font للـ Caption الخاص بـ label أخرى، والتى تعرض الـ font المختار.

فى غالبية الحالات، انك تريد ان تمكن المستخدمين من اختيار بنود الـ ListBox بضغط الـ Enter وقضيب المسافة. يقوم برنامج العرض ListFont بتشغيل هذه المفاتيح فى OnKeyDown الخاص بالـ FontList. يفحص الـ procedure ما اذا كان الـ Key موجود فى المجموعة ذات القيمتين VK_RETURN و VK_SPACE ، اذا كان كذلك، يستدع الـ FontListBoxDblClick. هذا عندماتضغط الـ Enter أو قضيب المسافة.

ان استخدام عامل in الخاص بالـ Pascal ومجموعة من القيم يعتبر اكثر فعالية، من تعبيرات الـ OR المنطقية المتعددة . على سبيل المثال، لاحظ الآتى:

```
if (Key = VK_RETURN) or
   (Key = VK_SPACE   ) or
   (Key = VK_F9        ) then {...}
```

بدلاً من العبارات السابقة، يمكنك اختبار ما اذا كان الـ Key فى مجموعة من ثلاث قيم بهذه العبارة :

```
if Key in [VK_RETURN, VK_SPACE, VK_F9] then {...}
```

اختيار بيانات list:

فى بعض الـ ListBoxes، تريد ان تمكن المستخدمين من ان يختاروا عنصر واحد فقط فى المرة. فى الـ ListBoxes الأخرى، تريد ان تمكنهم من اختيارات متعددة. ان اختيار عنصر واحد امر سهل- عليك فقط ان تستخدم مواصفات الـ ListBox الإفتراضية مع خاصية الـ MultiSelect المحددة بـ False. لتشغيل الاختيار متعدد البنود، حدد الـ MultiSelect بـ True.

بالإضافة للـ MultiSelect، حدد الـ ExtendedSelect بـ True أو False، يكون الـ ExtendedSelect ليس له تأثير (يمكنك ان تتركه محدداً بقيمة الـ

True الاصليـــة) ولكن عندمـــا يكون الـ MultiSelect مـــحـــدد بـ True، الـ ExtendedSelect يحدد اختياراً متعدد البنود بواحدة من طريقتين :

- **ExtendedSelect = True**: يجب ان يضـغط المسـتـخـدمـون Ctrl ويضغطوا الفأرة لاختيار عناصر متعددة.

بالإضافة الى ذلك، فإن ضغط Shift يختار كل العناصر الواقعة بين السطر الذى تم اختياره سابقاً والسطر الذى تم ضغطه. ان ضـغط Ctrl+Shift وضـغط الفأرة يفعل نفس الشئ ويحجز ايه بنود اخرى يتم اختيارها بشكل فردى، وضـغط عنصر مختار يؤدى الى إبطال اختياره.

- **ExtendedSelect = False**: لا يجب على المستخدمين ان يضغطوا ايه مفاتيح لاختياربنود متعددة، ان اختيار عنصر جديد لا يزيد الإبراز عن الاختيارات السابقة.

يمكنك تعـيين قـيم الخصـائص هـذه فى نافـذة الـ Object Inspector أو فى وقت التشغيل بعبارات مثل هذه :

```
ListBox1.MultiSelect := True;
ListBox1.ExtendedSelect := False;
```

عند تحديد الـ MultiSelect بـ True > والـ ExtendedSelect بـ False، تصـبح مـسـئـوليتك ان تزيل الاخـتـيـارات عن البنود المختـارة- عندمـا يضـغط المسـتخدمـون Esc ، مثلاً. والطريقـة البسيطة لفعل هذا هى بربط خـاصيـة الـ ExtendedSelect فتحاً وإغلاقاً كما توضح القائمة (٨-٣) استـخدم هذا الـ code فى OnKeyDown الخاص بالـ ListBox.

عند تحديد الـ MultiSelect بـ True، تشير خاصية الـ SelCount الى عدد العناصر المختارة. اذا لم يكن هناك بنود مختارة، يساوى الـ SelCount صفراً. عند تحديد الـ ExtendedSelect بـ False، استخدم OnKeyDown هذا لإبطال إبراز البنود المختارة فى ListBox عندما يضغط المستخدمون Esc .

القائمة (٨-٣): OnKeyDown لإبطال إبراز العناصر المختارة

```
procedure TForm1.ListBox1KeyDown(Sender: TObject;
```

```
      var Key: Word; Shift: TShiftState);
   begin
    if Key = VK_ESCAPE then
        with ListBox1 do
        begin
          ExtendedSelect := True;
              ExtendedSelect := False;
      end;
     end;
```

ان الـ SelCount دائماً يساوى 1- اذا كان الـ ExtendedSelect محدد بـ False، بغض النظر عما اذا كان البند مختاراً. عندما يكون الـ MultiSelect محدد بـ True، يشير الـ ItemIndex الى احدث عنصر تم ضغطه بغض النظر عما اذا كان هذا العنصر مختاراً أو غير مختار. وهذه الحالات الشاذة قد تحيرك. للحصول على افضل النتائج، لاتستخدم الـ SelCount مع الـ ListBoxes احادية المداخل، ولا تستخدم الـ ItemIndex مع الـ ListBoxes متعددة، المداخل.

تقترح القائمة (٨-٤) احدى طرق الحصول على اختيارات متعددة من الـ ListBox. يجب ان يكون الـ MultiSelect محدد بـ True حتى تعمل هذه الـ code بشكل مناسب. لتجرب هذه التقنية، اتبع هذه الخطوات:

١- أضف أثنان من الـ ListBox على الـ form.

٢- أضف بعض الـ strings فى خاصية الـ Items للـ ListBox1. حدد خاصية الـ MultiSelect للـ ListBox1 بـ True.

٣- أضف Button على الـ form واستخدام الـ list فى إنشاء الـ OnClick الخاص بالـ Button.

٤- قم بتشغيل البرنامج واضغط الزر لنقل البنود المختارة من الـ ListBox1 الى الـ ListBox2.

اختيار عناصر الـ ListBox فى وقت التشغيل:

فى بعض الاحيان يكون اختيار عناصر الـ ListBox باستخدام عبارات البرنامج أمراً نافعاً- على سبيل المثال، لإستعادة نافذة برنامج الى حالة الحفظ أو

لعمل اختيارات سابقة التحديد من مجموعات بيانات. ان ضغط الزر A قد يؤدى الى اختيـار بنود 3، 6، و 9 تلقائياً؛ وضغط الزر B يختـار بنود 2، و 7؛ الى آخره.

القائمة (٨-٤): الحصول على اختيارات متعددة من الـ list box

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  ListBox2.Clear;
     for I := 0 to ListBox1.Items.Count _ 1 do
       if ListBox1.Selected[I] then
          ListBox2.Items.Add(ListBox1.Items[I]);
end;
```

لإختيـار بنود فى ListBox آحادى المدخل فى وقت التشـغيل، عين قيـمة لخاصية الـ ItemIndex. على سبيل المثال، تختار العبارة التالية البند السادس (إن فهرس البند الأول هو صفر، لذا فإن فهرس البند السادس يساوى خمسة):

```
ListBox1.ItemIndex := 5;
```

ولكن هذا لا يعمل عندما تكون الـ MultiSelect محددة بـ True. فى هذه الحالة، لإختيار عناصر فى وقت التشـغيل، عين True لمدخـلات الـ Selected array. فمثلاً، تختار العبارتان التاليتان البند الثالث والسادس فى الـ ListBox1. يقوم Delphi تلقائياً بتعديل الـ SelCount والـ ItemIndex للقيم السليمة التالية لكل تعيين:

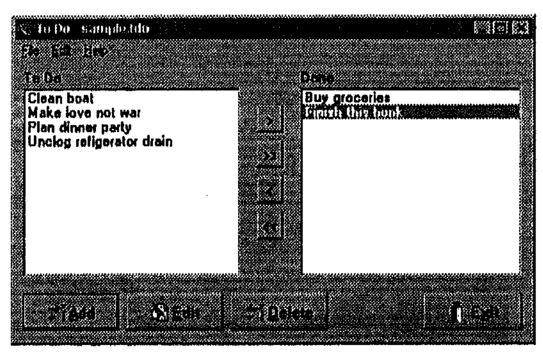
```
ListBox1.Selected[2] := True;
ListBox1.Selected[5] := True;
```

لأن Selected array لا يمكن التوصل إليه عندما تكون الـ MultiSelect محددة بـ False، فقد تريد استخدام عبارة if مثل التالية لإختيار البند السادس فى الـ ListBox1. يعمل هذا الـ code بشكل سليم للـ ListBoxes ذات المداخلات المتعددة والأحادية.

```
with ListBox1 do
if MultiSelect then
  Selected[5] := True
else
  ItemIndex := 5;
```

استخدام بيانات الـ ListBox:

كمثال عملي على استخدام كلا الـ ListBoxes البيانات فيما بينهما باستخدام الـ Button objects ، يعرض تطبيق الـ ToDo قائمة بسيطة للمشروعات التى يمكنك عملها . لقد استخدمت template من Dual list box الخاصة بـ Delphi لإنشاء نافذة البرنامج . ثم قمت بتعديل stock form لبرنامج الـ ToDo . يوضح شكل (٨-٢) عرض الـ ToDo . وخلال باقى هذا الباب ، سأوضح بعض code الـ ToDo ، والتى يصعب ذكرها هنا كاملة بسبب طولها .



شكل (٨-٢): يوضح تطبيق الـ ToDo كيفية استخدام template من Dual list box الخاص بـ Delphi لنقل البيانات بين الـ ListBox objects

تقدم template من Dual list box الـ code اللازم لنقل البيانات بين اثنين من الـ ListBox objects . لفحص هذا الـ code أضف from جديدة فى مشروع واختر مربع قائمة الـ Dual من template gallery . اعرض الـ form unit لترى كيف يتحكم Delphi فى كلاً من الـ ListBoxes .

إنشاء Sorted ListBoxes:

كما ذكرت ، إن إنشاء ListBoxes المرتبة يعد أمراً سهلاً- فقط حدد خاصية الـ Sorted بـ True . يحافظ الـ ListBox تلقائياً على ترتيب معين للإضافات

والمحــذوفــات . ولكن لأن الـ Windows هو الذى يؤدى التــرتيب ، وليس الـ Delphi component . على سبـيل المثـال ، بتـعـيين " Zebra" للبند الأول للـ ListBox1 :

```
ListBox1.Items[0] := 'Zebra';
```

لوضع الـ Zebra فى نهاية الحظيرة حيث ينتمى غالباً ، اجعل خاصية الـ Sorted مساوية الى on . هذه العبارات تجبر الـ ListBox أن يستعيد بياناته :

```
ListBox1.Sorted := False;
ListBox1.Sorted := True;
```

الـ ComboBoxes:

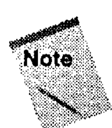
إن الـ ComboBoxes رائعة فى إنشاء قوائم إختيار ذات إمكانات عرض إخـتـيـارية . إن الـ ComboBox يدمج بين Edit Box (يشـبـه Edit الخـاص بـ Delphi) مع الـ ListBox . يستطيع المستخدمون إختيار بنود من القائمة ويمكنهم إدخـال بيـانات جـديدة فى نافـذة الـ edit control . وأنماط الـ ComboBox الأساسية الثلاثة هى :

- **Simple (Style = csSimple):** يكون الـ ListBox دائمـاً مـرئيـاً . يستطيع إدخال بند جديد فى نافذة الـ edit .

- **Drop-Down (Style = csDropDown):** يسـقط الـ ListBox عندما المستخدمون زر سهم الإسقاط المجاور لنافذة الـ edit– يمكنهم أيضاً ضغط Alt+Down . كما فى نمط الـ select ، يستطيع المستخدمون إختيار بند من القائمة ، أو يمكنهم إدخال عنصر جديد فى الـ edit . إن البحث المتزايد أيضاً قد تم تشغيله لهذا النمط– اكتب رمزاً واحداً أو أكثر واضغط مفتاح السهم العلوى أو السفلى لإختيار المدخل الذى يتماشى بشكل وثيق مع مدخلاتك .

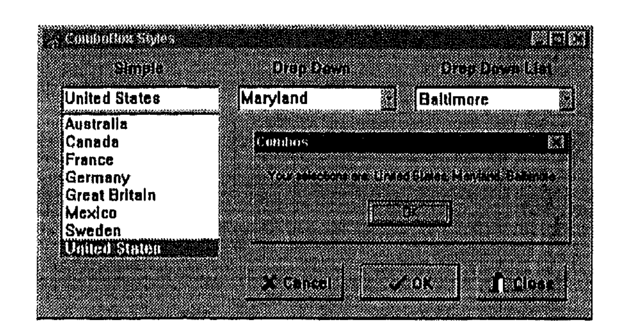
- **Drop-Down-List (Style = csDropDownList):** كما هو الحال فى نمط الـ Drop-Down ، يسقط الـ ListBox عند ضغط زر سهم الإسقاط . مع هذا الأسلوب الـ edit box سـيكون "read-only" ، ويجب أن يخـتـار المستخدمون مدخل مذكور فى القائمة . ولكن ، قد يكتب المستخدمون الحرف

الأول من البند لإختياره . على سبيل المثال ، اضغط مفتاح الـ P متكرراً لإختيار كل المداخل التى تبدأ بالـ P .



ملحوظة: على عكس الـ ListBox ، لا تستطيع الـ ComboBoxes عرض scroll bar أفقي . هذا الحد خاص بالـ Windows .

يوضح تطبيق الـ Combos أساليب الـ ComboBox الثلاثة . يوضح شكل (٨-٣) عرض البرنامج . توضح القائمة (٨-٥) الـ source file .



شكل (٨-٣): يوضح تطبيق الـ Combos أنماط الـ ComboBox الثلاث: Simple، Drop-Down، و Drop-Down-List

توضح قائمة الـ Combos الطريقة الصحيحة للحصول على إختيارات من الـ ComboBox . فى غالبية الحالات ، يجب أن تفعل هذا بالإشارة الى خاصية الـ Text للـ object والتى تحمل النص المعروض فى نافذة edit control . قد تستخدم الـ ListBox method للحصول على إختيارات ، مثلاً ، بالإشارة الى الـ Items array :

```
if ItemIndex >= 0 then
  S1 := SimpleCB.Items[ItemIndex];  // ???
```

القائمة (٨-٥): Combos\Main.pas

```
unit Main;
interface
uses
  SysUtils, Windows, Messages, Classes, Graphics,
```

```
Controls, Forms, Dialogs, StdCtrls, Buttons;

type
  TMainForm = class(TForm)
    SimpleCB: TComboBox;
    DropDownCB: TComboBox;
    DropDownListCB: TComboBox;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    CancelBitBtn: TBitBtn;
    OKBitBtn: TBitBtn;
    CloseBitBtn: TBitBtn;
    procedure CancelBitBtnClick(Sender: TObject);
    procedure OKBitBtnClick(Sender: TObject);
    procedure CloseBitBtnClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  MainForm: TMainForm;
implementation
{$R *.DFM}

procedure TMainForm.CancelBitBtnClick(Sender: TObject);
begin
  Close;
end;

procedure TMainForm.OKBitBtnClick(Sender: TObject);
```

```
var
    S1, S2, S3: string;
begin
    S1 := SimpleCB.Text;
    S2 := DropDownCB.Text;
    S3 := DropDownListCB.Text;
    ShowMessage('Your selections are: ' +
     S1 + ', ' + S2 + ', ' + S3);
  end;

  procedure TMainForm.CloseBitBtnClick(Sender: TObject);
  begin
   OKBitBtn.Click;  { Simulate OK button click }
     Close;           { End program }
  end;

  end.
```

وهذه التقنية موضع استفسار لأنها لا تنتقى بشكل سليم مداخل المستخدم من حقل الـ Text . للحصول على أفضل النتائج، احضر الإختيارات من خاصية الـ Text إلا اذا كنت تحتاج الى أداء أعمال على المداخل فى نافذة الـ ListBox. على سبيل المثال، تنسخ هذه العبارة الإختيار من الـ SimpleCB ComboBox الى متغير S1 :

```
S1 := SimpleCB.Text;
```

فى وقت التصميم، أدخل بنود قائمة فى خاصية الـ Items للـ ComboBox. أدخل أو امسح خاصية الـ Text لتغيير مساحة الـ string edit، والتى يعين لها Delphi اسم الـ object حسب النظام الإفتراضى.

الـ ComboBox يقدم method أخرى قد تجدها نافعة فى تصميم واجهة تطبيق برنامجك. استخدم خاصية الـ DroppedDown لتحدد ما اذا كان الـ ComboBox معلق الى أسفل :

```
if DropDownCB.DroppedDown then
  {... do something if ComboBox is open }
```

عين بـ True أو False للـ DroppedDown لفتح أو إغلاق جزئية القائمة للـ ComboBox الواقعة تحت البرنامج . على سبيل المثال، أضف ComboBox و Button على الـ form، واستخدم هذه العبارة OnClick الخاص بالـ Button لفتح وإغلاق نافذة قائمة الـ ComboBox :

```
with ComboBox1 do
  DroppedDown := not DroppedDown;
```

لإختيار كل الـ text في مساحة الـ ComboBox's edit، استدع الـ SelectAll method . هذا يبرز نص الـ edit- إنه لا يختار كل بنود القائمة كما يبدو من اسم الـ method. على سبيل المثال، يمكنك استخدام هذه العبارة في OnKeyDown لإبراز نص الـ edit عندما يضغط المستخدمون مفتاحاً بعينه :

```
SimpleCB.SelectAll;
```

إن الـ ComboBoxes هي objects آحـادية المدخل . إن الـ ListBox components فقط هي التي تتعامل مع الإختيارات المتعددة .

إن استدعاء الـ Clear يمسح كل مداخل القائمة وأي نص في edit box من الـ ComboBox object . لحذف كل النص من الـ ComboBox، استخدم هذه العبارة :

```
ComboBox1.Clear;
```

اذا أردت حذف النص فقط من الـ edit، عين null string لخاصية الـ Text للـ ComboBox :

```
ComboBox1.Text := '';
```

يمكنك أيضاً استخدام الـ methods الموضحة سابقاً لعمل التغييرات ببنود القائمة من خلال الـ Items array. فمثلاً، هذا يضيف مدخل جديد لمربع الـ ComboBox's list :

```
ComboBox1.Items.Add('Aardvark');
```

:String and Other Lists

تعتبر الـ ListBox والـ ComboBox أدوات بصرية لإنشاء قوائم إختيار بالنوافذ . تحتاج غالبية البرامج أيضاً الى إمكانات صنع قوائم داخلية باستخدام



objects ليست فى لوحة الـ VCL الخاصة بـ Delphi. يوضح هذا الفصل ثلاث classes يمكنك استخدامها للتحكم فى string list الـ list الأخرى فى تطبيقاتك. حسب الترتيب الأبجدى، الـ classes الثلاث هى:

- **TList:** class لصنع قائمة.

- **TStringList:** وهى class يمكن للتطبيقات استخدامها لإنشاء قوائم للـ string والـ objects. تحدد الـ TStringList الذاكرة اللازمة لتخزين الـ string. استخدم هذا النوع لإنشاء متغيرات مستقلة عندما تحتاج مكاناً لتخزين قوائم الـ Strings.

- **TStrings:** وهى abstract class تستخدمها خصائص الـ component للتحكم فى قوائم الـ strings الـ objects الأخرى. لا تقوم الـ TStrings objects بتخصيص أية ذاكرة للتخزين. إن المرة الوحيدة التى تستخدم هذا النوع من القوائم هى عندما تشير الى خصائص الـ component object. إنك لا تنشئ أبداً متغيرات مستقلة من هذا النوع.

Note **ملحوظة:** إن الـ classes السابقة ليست أدوات واجهة تطبيق بالمعنى المفهوم، ولكننى أريد أن أقدمها هنا فى الجزء الثانى من الكتاب بسبب أهميتها البالغة فى تطوير تطبيقات Delphi. ولكن، إننى أشرح أيضاً الـ StringGrid component، والذى يستخدم قوائم الـ string وهو أداة قيمة جداً لإنشاء واجهات تطبيق عملية للمستخدم.

استخدام الـ TList class:

إن Delphi يستخدم الـ TList class كأساس لإنشاء قوائم ذات أغراض عامة. يمكنك أيضاً استخدام الـ TList كـ class لتخزين الـ objects من أى نوع فى قوائم. فى أغلب الحالات، لكى تستخدم الـ TList، إنك تحتاج لإنشاء إثنين من الـ class- واحدة لبنود الـ objects للتخزين فى القائمة، وواحدة للقائمة ذاتها. على سبيل المثال، يمكنك إنشاء class فى جزئية تنفيذ unit module. إن العناصر الموجودة فى القائمة هى objects من نوع الـ class الخاص بك، والتى يمكن تعريفها فى واجهة التطبيق للوحدة مثلما يلى:

```
type
 TAnyItem = class
     Data: Integer;
       constructor Create(Data1: Integer);
      end;
```

كما هو مقرر هنا، تعتبر الـ TAnyItem نوع بيانات class توفر متغير عدد صحيح يسمى Data. بالرغم من عدم الاستخدام العملى، تظهر الـ TAnyItem الاسس لإنشاء قائمة للـ objects. بالإضافة الى عضو الـ Data التابع لها، فإن الـclass لها constructor يسمى Create لبدء تكوين الـ objects بالـ class. اجعل دائماً اسم constructor هو Create. نفذ الـ constructor كما تفعل مع أى procedure، ولكن استخدام كلمة الأساسية constructor فى مكان الـ procedure، وإسبق اسم الـ method بـ TAnyItem ونقطة. ما يلى يوجد فى قطاع الـ implementation:

```
constructor TAnyItem.Create(Data1: Integer);
begin
    Data := Data1;
    inherited Create;
end;
```

يحفظ الـ constructor parameter الـ Data1 فى متغير الـ Data الخاصة بالـobject ثم يستدعى الـ Create object المشتقة من الـ ancestor class. إن استدعاء الـ Create المشتق يعطى الـ ancestor class الفرصة لأداء البدء الخاص بها، أى كانت. (فى البرمجة المختصة بالـ object، إننا لا نحتاج أن نعرف كل التفاصيل الدقيقة عن ancestor class methods). فى هذه الحالة، الـ ancestor class هى الـ TObject، الـ class الأم لكل Delphi classes، حتى تلك التى تنشئها أنت بنفسك.

ملحوظة: اعتبر الـ class constructor كمنشئ وللعناصر الداخلية لـ object جديد. لكل object من الـ class، يجب ان يستدعى البرنامج الـ Create constructor لبدء الـobject.

إنك تحتاج ايضاً class للقائمة . لا يوجد سبب وجيه لإعادة ابتكار class ، ان تستغل الـ methods التى توفرها الـ TList ، ان تستخرج الـ class الجديدة من الـ TList . ترث الـ class الجديدة الخصائص والـ methods من الـ TList . بعبارة اخرى ، ان الـ class الجديدة هى نفسها الـ TList ، زائد أى إمكانات تقرر إضافتها . لإنشاء class مشتقة ، استخدم هذا التعريف قطاع فى الـ unit's implementation :

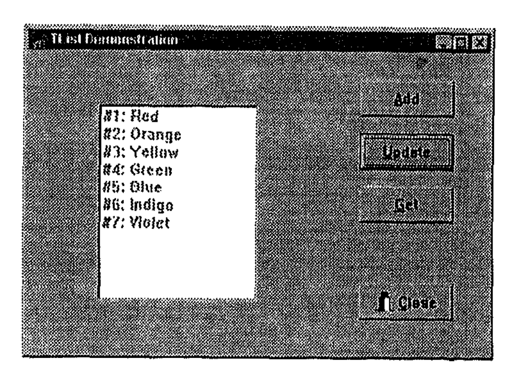
```
type
  TAnyList = class(TList)
      destructor Destroy; override;
      end;
```

الـ TAnyList class تستخدم الـ override للـ destructor Destroy فى الـ ancestor TList class . وهذا ضرورى لان الـ TList objects فى العادى لا تدمر الـ objects التى تحتوها . لإضافة هذه الامكانية للـ class المشتقة ، قم بتنفيذ الـ destructor من خلال قطاع unit's implementation ، كما يلى :

```
destructor TAnyList.Destroy;
var
  I: Integer;
begin
     for I := 0 to Count - 1 do
       TAnyItem(Items[I]).Free;
        inherited Destroy;
end;
```

أولاً ، تستدعى for loop الـ Free لكل الـ TAnyItem objects على القائمة . تحافظ الـ TList class على بنود الـ Count . تحتوى خاصية الـ Items على pointer تشير الى (عناوين) الـ objects فى القائمة . لإستدعاء method مثل الـ Free يتطلب تعبير type-cast كما هو موضح فى هذا المثال ، والذى يخبر الـ compiler إن الـ Items[I] ليس مجرد pointer ، ولكن الـ TAnyItem object .
أخيراً ، يستدعى الـ destructor الجديد الـ Destroy المشتق لإعطاء الـ TList class فرصة لأداء المسح الخاص به .

يمكنك ، وأنت مسلح بهذه الأسس ، أن تستخدم الـ TList لإنشاء قوائم من أى نوع من الـ objects . يظهر برنامج الـ StrList الخطوات الازمة . يوضح شكل (٨-٤) عرض البرنامج . تعطى القائمة (٨-٦) الـ source code . قم بتشغيل البرنامج ، اضغط Add ، وإدخل string . أعد هذه الخطوات عدة مرات لإنشاء قائمة string objects فى الذاكرة .



شكل (٨-٤) : يوضح تطبيق الـ StrList كيفية استخدام الـ TList class بـ Delphi's لإنشاء قوائم من الـ objects فى الذاكرة

اضغط Update لنقل objects القائمة الى الـ ListBox . (لن ترى الـ strings الخاص بك حتى بعد ان تضغط الـ Update) . اضغط Get وإدخل strings للبحث فى القائمة . سوف اشرح المزيد عن الـ code بعد القائمة .

القائمة (٨-٦) : Strlist\Main.pas

```
unit Main;

interface

uses
  SysUtils, Windows, Messages, Classes, Graphics,
     Controls, Forms, Dialogs, StdCtrls, Buttons;

type
  TMainForm = class(TForm)
     AddButton: TButton;
```

```
      CloseBitBtn: TBitBtn;
        GetButton: TButton;
        ListBox1: TListBox;
        UpdateButton: TButton;
        procedure FormCreate(Sender: TObject);
        procedure FormDestroy(Sender: TObject);
        procedure AddButtonClick(Sender: TObject);
        procedure UpdateButtonClick(Sender: TObject);
        procedure GetButtonClick(Sender: TObject);
      private
    { Private declarations }
     public
    { Public declarations }
     end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}

type

{- New class of items to insert in a list }
  TStrItem = class
       S: string;
        I: Integer;
        constructor Create(S1: string; I1: Integer);
 end;
   {- Derived class to hold TStrItem objects }
    TStrList = class(TList)
        destructor Destroy; override;
```

```
        function FindItem(S1: string): TStrItem;
         end;

var
 StrList: TStrList;        { List of TStrItems }
      ItemCount: Integer;     { Number of items inserted }

{- Create a new instance of TStrItem }
constructor TStrItem.Create(S1: String; I1: Integer);
begin
 S := S1;              { Save string parameter }
      I := I1;               { Save integer parameter }
      inherited Create;      { Call inherited Create }
  end;

  {- Destroy instance of TStrList }
  destructor TStrList.Destroy;
  var
   I: Integer;
  begin
   for I := 0 to Count - 1 do
       TStrItem(Items[I]).Free;  { Free all TStrItems }
       inherited Destroy;        { Call inherited destroy }
  end;

  {- Return object identified by S1 or nil for no match }
  function TStrList.FindItem(S1: string): TStrItem;
  var
      I: Integer;
      P: TStrItem;
  begin
   for I := 0 to Count - 1 do
      begin
```

```
    P := TStrItem(Items[I]);  { P refers to a TStrItem object }
      if Uppercase(P.S) = Uppercase(S1) then  { Match? }
      begin          { Found match }
        Result := P;  { Return P as function result }
          Exit;        { Exit function immediately }
        end;
      end;
    Result := nil;    { No match; return nil }
end;
  {- TMainForm event handlers }

  procedure TMainForm.FormCreate(Sender: TObject);
  begin
    StrList := TStrList.Create;  { Create new StrList object }
      ItemCount := 0;              { Initialize insertion count }
  end;
  procedure TMainForm.FormDestroy(Sender: TObject);
  begin
    StrList.Free;  { Also destroys listed items }
  end;
  {- Button event handlers }
  procedure TMainForm.AddButtonClick(Sender: TObject);
  var
    StrItem: TStrItem;   { New item to insert }
      S: string;          { User input string }
  begin
    S := '';
      if InputQuery(Caption, 'Enter item', S) then
      if Length(S) > 0 then
      begin
      Inc(ItemCount);
        StrItem := TStrItem.Create(S, ItemCount);
        StrList.Add(StrItem);
```

```
    end;
  end;

  procedure TMainForm.UpdateButtonClick(Sender: TObject);
  var
    I: Integer;
      P: TStrItem;
  begin
    ListBox1.Clear;
      with StrList do
      for I := 0 to Count - 1 do
      begin
        P := TStrItem(StrList.Items[I]);
          ListBox1.Items.Add(Format('#%d: %s', [P.I, P.S]));
      end;
  end;

  procedure TMainForm.GetButtonClick(Sender: TObject);
  var
    S: string;
      P: TStrItem;
  begin
    UpdateButton.Click;
      S := '';
      if InputQuery(Caption, 'Enter item name', S) then
      if Length(S) > 0 then
      begin
        P := StrList.FindItem(S);
          if P <> nil then
            ShowMessage(Format('%s, Number = %d', [P.S,
        P.I]))
            else
          ShowMessage('No such item');
        end;
```

```
  end;

end.
```

يظهر الـ StrList كيفية إنشاء قائمة مرتبطة، والتى تتطلب إثنان من الـ classes. الـ class الأولى، TStrItem، تعرف إثنتين من أعضاء البيانات- الـ string S، وقيمة العدد الصحيح I. تعرف الـ class أيضاً Create constructor لبدء هذه المتغيرات. والـ class الثانية، TStrList، والمتشقة من الـ TList، تعرف إثنان من الـ methods: destructor و function، FindItem، التى تبحث فى القائمة عن object معرف بمتغير string.

لإنشاء object القائمة، يعرف البرنامج متغيراً، وهو StrList، من الـ TStrList class. اذا كنت تفضل، يمكن أن تذهب هذه المتغيرات الى الـ form class.

الـ TStrItem constructor يعين الـ parameter عدد صحيح و string لمتغيرات الـ I والـ S الخاصة بالـ class. هذا يبدأ فى إدخال objects جديدة فى القائمة. لاحظ أن الـ constructor يستدعى الـ Create الموروث، بالرغم من أن الـ TStrItem تعد class جديدة ليست مشتقة من class أخرى. وهذه الخطوة ليست ضرورية، ولكن يجب أن تفعلها على أية حال لأن، فى Pascal الخاصة بـ Delphi، تعتبر كل الـ classes مشتقة من الـ class الأم TObject.

Tip **فكرة:** لأن الـ classes الجديدة تعتبر مشتقة من الـ TObject بشكل تلقائى، لذا فإن كل Delphi classes و objects تعتبر مرتبطة ببعضها. من بين المميزات الأخرى، هذا يعنى أنه بإمكانك تمرير object من أى نوع class الى الـ TObject parameter فى الـ procedure أو الـ function.

توفر الـ TObject، Create constructor إفتراضى و Destroy destructor إفتراضى، زائد code لإرسال رسالة. باختيار library source Delphi code وقت التشغيل، تجد تعريف Pascal الخاص بالـ TObject فى ملف الـ System.pas، الموجود فى المسار Source\Rtl\Sys وكما ستعرف بتصفح هذه المعلومة، إن الـ TObject methods مكتوبة باستخدام عبارات لغة الـ Assembly. ولقد جرت العادة على أن يتم تخزين هذه الـ code منفصلة فى

ملفات Clsf.asm و Clsh.asm؛ وهى الآن فى System.pas. (أنك لا تحتاج الـ source code لاستخدام الـ TObject، ولكن اذا قمت بكثير من البرمجة فى Delphi، فيمكنك معرفة الكثير بتصفح component methods مثل هذه).

إنك لا تحتاج الى اشتقاق classes من الـ TObject. على سبيل المثال، التعريفان للـ class التاليين يعتبران متساوين:

```
type
     TNewClass1 = class
       {...}
     end;
     TNewClass2 = class(TObject)
      {...}
      end;
```

ان الـ TStrList destructor، يستدعى Free لكل TStrItem object على القائمة. ثم يستدعى الـ Destroy الموروث. قد يستخدم code مشابهة فى method اخرى- مثلاً، واحداً يمسح الـ objects من القائمة.

تبحث الـ Function FindItem فى قائمة الـ objects عن objects يتماشى مع string parameter. تتكرر الـ for loop فى القائمة باستخدام الـ Count الخاص بالـ TList. لتبسيط الـ code، تعين العبارة كل objectsفى القائمة الى الـ P:

```
     P := TStrItem(Items[I]);
```

ان P هو متغير من list object class (TStrItem)، . ولكن بالرغم من المظاهر، ان الـ P لا يحتوى على بيانات حالة الـobject. يحتوى الـ P على pointer reference الى الـ TStrItem object instance. فى Delphi ، تعتبر كل متغيرات الـ class-object ، مثل الـ P، reference أى انها pointers تشير الى الـ objects فى الذاكرة. اذا كنت تعرف نسخ اخرى من الـ Pascal، قد يبدو الـ code السابق على انها تنسخ TStrItem object باكمله لـ P، ولكن هذا لا يحدث فى الـ Object Pascal بـ Delphi. ان كل ما يتم تعينه هو العنوان التابع الـTStrItem object فى الـ Items array بالقائمة. تعتبر كل متغيرات الـ class - أى الـ objects- تعتبر references.

داخل الـ for loop، تقارن عبارة if الـ object string (P.S) بالقيمة المتغيرة (S1) التي تم تمريرها الى الـ FindItem . تستخدم العبارة الـ Uppercase function بـ Delphi لمقارنة الـ strings بغض النظر عن الاحرف . اذا توافقت الـ strings، يعين البرنامج الـP للـ Result الخاصة بالوظيفة، ثم يخرج الـ function على الفور باستدعاء الـ Exit procedure بالـ Pascal . اذا لم يتوافقا، تعين العبارة النهائية لاشئ للـ Result الخاص بالـ function .

بالرغم من ان الـ function للـ FindItem هي TStrItem، تقـوم الـ function في الواقع بادخال reference الى الـ TStrItem object. هذه هي فائدة اخرى لقاعدة Delphi ان كل متغيرات الـ class-object تعتبر references .
وبالتالى، يسمح Delphi بتعين لاشئ للـ Result الخاص بالـ FindItem ليشير الى انه لا يرجع الى أى object .

يقوم OnCreate الخاص بالـ StrList للـ form بانشاء قائمة جديدة بتنفيذ العبارة التالية، والتى تستدعى الـ Create method الخاص بالـ TStrList . وتقوم عبارة تعيين بانشاء TStrList object في الذاكرة، وتبدأ متغير الـ TStrList ليعود على للـ object :

```
StrList := TStrList.Create;
```

ان احدى قواعد Delphi الاساسية هي انك يجب ان تمحو الـ objects التي تنشئها . في هذه الحالة OnDestroy بتحرير object القائمة باستدعاء الـ Free method الموروث من الـ TList :

```
StrList.Free;
```

ولان الـ TStrList destructor يمحو كل objects القائمة بشكل سليم يتخلص من كل الذاكرة المخصصة للقائمة والـ objects التي تحتوى عليها .

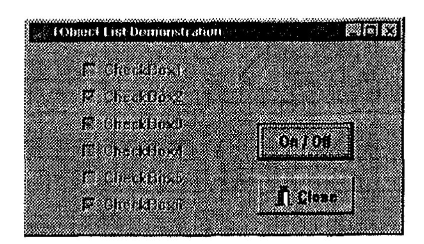
لإضافة objects جديدة للقائمة، فإن الـ OnClick الخاص بالـ AddButton's يحثك على إضافة string تنشئ هاتان العبارتان TStrItem object جديد وتضيفاه للقائمة باستدعاء Add method للـ TList :

```
StrItem := TStrItem.Create(S, ItemCount);
StrList.Add(StrItem);
```

فى البـرنامج المثـال، عندمـا تضـغط زر الـ Update ، يقـوم الـ OnClick الخاص به بنقل كل الـ string والقيمة العددية الصحيحة الخاصة بالـ object الى الـ ListBox component object. يرجع التـعـبـيـر P.I و P.S على البـيـانات الموجودة فى كل TStrItem object فى القائمة . وتعرض تعبيرات مشابهة فى الـ GetButtonClick اعداد صحيحة و object strings بعد استدعاء الـ FindItem للبحث فى القائمة .

إنشاء Array of TObject References

فى التطبيـقـات المـعـقـدة، من الصـواب ان تنشئ array من الـ TObject references لـ components المتنوعة فى الـ form. ان كل الـ classes تنحدر من الـ TObject، ولذلك، فإن الـ TObject array يمكنها ان تعود على اى Delphi object. على سبيل المثال، يمكنك إدخال مجموعة من component objects فى الـ TObject array، ثم تستخدم الـ array لاداء عمليات على كل الـ objects. يوضح تطبيق الـ ObjList [انظر القائمة (٨-٧)] التقنيات اللازمة. وتحتوى نافذة الـ form على ستة من الـ CheckBox objects. اضغط زر الـ On/Off الخاص بالـ form لابطال وتشغيل الـ CheckBoxes. ينفذ البـرنامج هذه المهمة بتعـريف pointer الى الـ TObject array، ثم استخدام الـ GetMem لتخصيص ذاكرة لعد الـ CheckBoxes بالـ form. يقـوم OnCreate الخاص بالـ form بتعيين كل الـ CheckBoxs الى عناصـر الـ CheckBoxArray (تذكر ان جميعها references). ويستخدم الـ OnClick الخاص بالزر for loop لكى يجعل كل خاصية Enabled لـ CheckBox object بـ off او on. إخيرا، يستدعى OnDestroy الخاص بالـ form الـ FreeMem للتخلص من الذاكرة المخصصة. يوضح شكل (٨-٥) عرض البرنامج.



شكل (٨-٥): يوضح الـ ObjList كيفية إنشاء array لـ TObject references لاداء عمليات على مجموعات الـ objects

القائمة (٨-٧) : الـ Objlist\Main.pas

```
unit Main;

interface

uses
    SysUtils, WinTypes, WinProcs, Messages, Classes,
  Graphics,
    Controls, Forms, Dialogs, StdCtrls, Buttons;

type
 TMainForm = class(TForm)
      CheckBox1: TCheckBox;
      CheckBox2: TCheckBox;
      CheckBox3: TCheckBox;
      CheckBox4: TCheckBox;
      CheckBox5: TCheckBox;
      CheckBox6: TCheckBox;
      OnOffButton: TButton;
      BitBtn1: TBitBtn;
      procedure FormCreate(Sender: TObject);
      procedure FormDestroy(Sender: TObject);
      procedure OnOffButtonClick(Sender: TObject);
    private
     { Private declarations }
     public
     { Public declarations }
     end;

var
    MainForm: TMainForm;

implementation
```

```
{$R *.DFM}

const
 numCheckBoxes = 6;

type
 PObjectArray = ^TObjectArray;
    TObjectArray =
      array[0..65520 div SizeOf(TObject)] of TObject;

var
    CheckBoxArray: PObjectArray; //Pointer to array of checkBoxes

procedure TMainForm.FormCreate(Sender: TObject);
begin
    GetMem(CheckBoxArray, numCheckBoxes * SizeOf
   (TObject));
    CheckBoxArray^[0] := CheckBox1;
    CheckBoxArray^[1] := CheckBox2;
    CheckBoxArray^[2] := CheckBox3;
    CheckBoxArray^[3] := CheckBox4;
    CheckBoxArray^[4] := CheckBox5;
    CheckBoxArray^[5] := CheckBox6;
end;

procedure TMainForm.FormDestroy(Sender: TObject);
begin
 FreeMem(CheckBoxArray, numCheckBoxes * SizeOf(TObject));
end;

procedure TMainForm.OnOffButtonClick(Sender: TObject);
var
 I: Integer;
begin
```

```
    for I := 0 to numCheckBoxes - 1 do
      with CheckBoxArray^[I] as TCheckBox do
        Enabled := not Enabled;
  end;

    end.
```

استخدام الـ TStringList class:

Delphi يقدم class اخرى وهى الـ ,TStringList التى تستخدم الـ TList object لإنشاء string lists . تنحدر الـ TStringList من الـ TStrings، وهى الـ class التى تستخدمها الـ components لخصائص الـ string-list (سوف اشرح المزيد عن الـ TStrings فى الفصل القادم). باستخدام الـ TStringList، يمكنك إنشاء lists مرتبطة للـ strings والـ objects الأخرى، ويمكنك تعيين تلك القوائم لخاصية الـ TStrings فى أى .Delphi component object .

ان واحدة من اشهر استخدامات الـ TStringList هى قراءة وكتابة ملفات النص . فى الماضى البعيد، لان الـ TStringList كانت تستخدم الـ Pascal strings، لم تكن السطور اطول من ٢٥٥ رمزاً. لم يعد هذا الحد موجوداً الآن، فالـ Object Pascal Strings يمكن ان تكون بأى طول. توضح القائمة (٨-٨) كيفية استخدام الـ TStringList لانشاء ملف نص جديد. لتجربة الـ code ، قم بانشاء مشروع وأضف إثنين من الـ Button objects على الـ form. وكذلك أضف الـ Memo component. إدخل العبارات من القائمة (٨-٨) فى OnClick الخاص بالـ Button1.

القائمة (٨-٨): يوضح OnClick كيفية استخدام الـ TStringList لإنشاء ملف نص

```
procedure TForm1.Button1Click(Sender: TObject);
var
  SL: TStringList;
begin
    SL := TStringList.Create;
    try
```

```
    SL.Add('This is the first line');
      SL.Add('This is the second line');
      SL.Add('This is THE END');
      SL.SaveToFile('Anyname.txt');
   finally
    SL.Free;
     end;
end;
```

يقوم البرنامج أولاً بإنشاء الـ TStringList يسمى SL . ثم يضيف بعض الـ strings الى القـائمة، ويستـدعى الـ SaveToFile اكتـابتهـا فى ملف نص يسـمى Anyname.txt . يقـوم الـ procedure باسـتـدعـاء Free لحـذف الـ string list object من الذاكـرة . دائمـاً أمـحـو الـ objects التى تنشـئهـا . ويضـمـن عـبـارة الـ try-finally-end ان الـ SL تم محوه، حتى لو كـانت أى عـبارة بعد الـ try تسبب exception .

توضح القائمة (٨-٩) كيفية قراءة ملف نص فى الـ TStringList . اذا كنت تنشئ المشروع، استخدم هذا الـcode فى OnClick الخاص بالـ Button2 . يقوم الـ procedure بإنشـاء الـ SL object للـ TStringList، ثـم يسـتـدعى الـ LoadFromFile لقـراءة سطور من الـ Anytext.txt فى list object . ويقـوم بتعيين تلك النسخ من البيانات الى خاصية الـ Lines لـ Memo1 object's، وهو object من نوع الـ TStrings .

لان الـ TStringList تنحـــدر من الـ TStrings، يمكنك تعـــيين أى TStringList object لخـاصـيـة الـ TStrings لـ component object's . يمكنك ايضاً تعيين ايه خاصية TStrings لـ TStringList object .

القائمة (٨-٩): يوضح OnClick هذا كيفية استخدام الـ TStringList object لقراءة ملف نص

```
procedure TForm1.Button2Click(Sender: TObject);
var
  SL: TStringList;
begin
```

```
SL := TStringList.Create;
 try
    SL.LoadFromFile('Anyname.txt');
    Memo1.Lines := SL;
 finally
  SL.Free;
   end;
end;
```

توضح القائمة (٨-٩) كيفية قراءة ملف نص فى TStringList object مستقل، والذى قد تريد فعله لكثير من اغراض التعامل مع القوائم. ولكن، يمكنك ببساطة اكثر ان تقوم بتحميل خاصية الـ Lines لـ Memo component باستدعاء الـ LoadFromFile method الخاص بها. فهذه العبارة تفعل كل ما تفعله القائمة (٨-٩):

```
Memo1.Lines.LoadFromFile('Anyname.txt');
```

استخدام الـ AddStrings لإلحاق string lists متعددة الـTStringList object أو خاصية الـ TStrings. على سبيل المثال اذا لديك ثلاثة من الـ TStringList -T1، T2، T3- يمكنك إدخالها فى خاصية الـ Lines لـ Memo object بواسطة العبارة التالية. يحل التعيين الاول محل نص الـ Memo بـ strings من الـ T1. والتعيين الآخرين يلحقا القائمتين الاخرتين:

```
with Memo1, Lines do
begin
 Lines := T1;
   AddStrings(T2);
   AddStrings(T3);
end;
```

TStringList لابد من تحديد قيمة لها، ليس فقط string lists ، ولكن ايضاً لإنشاء روابط بين الـ strings والـ objects. يوضح الفصل القادم كيفية استخدام هذه الإمكانية لعرض bitmaps فى الـ ListBox control.

استخدام الـ TStrings class :

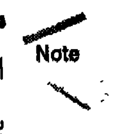
تعتبر الـ alstract class، وهذا يعنى ان البرامج لا تستطيع إنشاء objects من هذا النوع. قد تستخدم الـ TStrings فقط كخاصية component object .

مثلاً، يمكنك تخزين قائمة من الـ strings والـ objects الاخرى فى خاصية الـ Items التابعة للـ ListBox's، وهو TStrings object. لإنشاء list objects الخاصة بك، استخدم الـ TList والـ TStringList classes كما هو موضح فى الفصول السابقة.

لان الـ TStringList تنحدر من الـ TStrings، فإن المعلومات الواردة فى هذا الباب تنطبق على كلتا الـ classes. فأى شئ يمكنك فعله مع خاصية الـ TStrings، يمكنك فعله ايضاً مع الـ TStringList object.

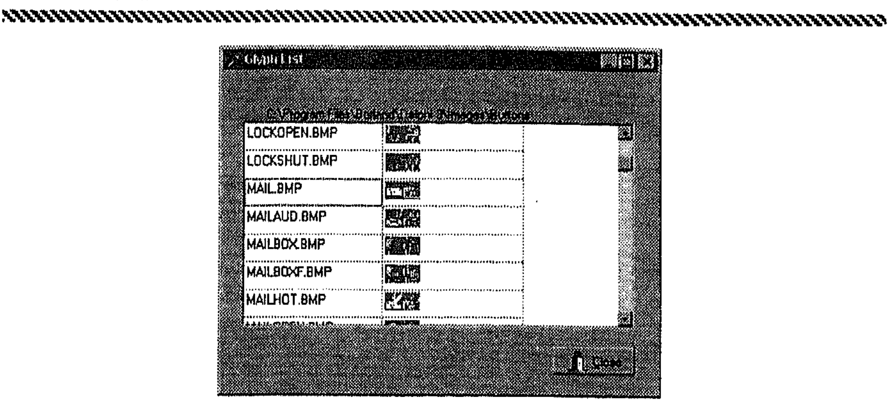
استخدام الـ StringGrid :

يعد الـ StringGrid مربع list box متقدم قادراً على ربط الـ string والـ object الاخرى مثل الـ bitmap. يمكن ان يعرض الـ StringGrid جداول لبيانات الـ strings، كلاً منها مرتبط بـ object آخر.



على القرص المدمج: يوضح البرنامج التالى، GlyphLst كيفية استخدام الـ StringGrid لإنشاء array of strings . والـ bitmaps يعرض البرنامج كل glyph bitmaps الخاصة بـ Delphi واسماء الملفات الخاصة بها فى owner-draw string grid object . (انظر الباب السادس لمزيد من المعلومات عن الـ glyphs). والمصطلح owner-draw يعنى أن البرنامج، بدلاً من الـ Windows، يتولى مسئولية رسم كل عنصر فى خلية الـ StringGrid . يوضح شكل (٨-٦) عرض الـ GlyphLst. توجد الـ source code فى القائمة (٨-١٠). يمكنك أن تجد ملفات البرنامج على القرص المدمج فى دليل الـ Source\GlyphLst. لقد أدخلت تعليقات فى الـ code لتوضيح كثير من العبارات، التى تستخدم بعض التقنيات لم يتم تقديمها بعد.

ملحوظة: اذا لم تقم بتركيب Delphi فى الأدلة الإفتراضية على محرك الـ C:، قم بتعديل string اسم المسار فى ثابت الـ glyphPath، الموضوع بعد كلمة التنفيذ الرئيسية للـ unit مباشرة.



شكل (٨-٦): يعرض الـ GlyphLst الـ owner-draw ListBox لكل bitmaps الموجودة مع Delphi

القائمة (٨-١٠): Glyphlst\Main.pas

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics,
 Controls,
  Forms, Dialogs, StdCtrls, Buttons, Grids;

type
  TMainForm = class(TForm)
   PathLabel: TLabel;
    BitBtn1: TBitBtn;
    GlyphList: TStringGrid;
    procedure FormCreate(Sender: TObject);
    procedure FormClose(Sender: TObject;
     var Action: TCloseAction);
      procedure GlyphListDrawCell(Sender: TObject;
     Col, Row: Integer; Rect: TRect; State: TGridDrawState);
     private
   { Private declarations }
```

```
    public
      { Public declarations }
       end;

var
    MainForm: TMainForm;

implementation
{$R *.DFM}

const glyphPath =
 'C:\Program Files\Borland\Delphi 4\Images\Buttons';

// Reads all glyph bitmaps and stores their filenames and
// images in the GlyphList TStringGrid object.
procedure TMainForm.FormCreate(Sender: TObject);
var
 SearchRec: TSearchRec;  { Directory scan result record }
    K: Integer;           { while-loop control variable }
    Bitmap: TBitmap;       { Holds bitmaps. Do not Free! }
    Index: Integer;        { TStringGrid cell index }
begin
    Show;  { Make form visible while loading bitmaps }
    Screen.Cursor := crHourGlass;  { Show hourglass cursor }
    Index := 0;
    try
     PathLabel.Caption := glyphPath; { Show path above
     ListBox }
        { Start scan }
        K := FindFirst(glyphPath + '\*.*', faAnyFile,
     SearchRec);
        try
         while K = 0 do  { Scan directory for file names }
      begin
```

```
if SearchRec.Name[1] <> '.' then {No '.' or '..' paths}
    begin
     Bitmap := TBitmap.Create;  { Create bitmap object }
        try  { Get bitmap and load from list }
            Bitmap.LoadFromFile(glyphPath + '\'
             + SearchRec.Name);
            if Index = GlyphList.RowCount then  // Expand
list
             GlyphList.RowCount := Index + 1;
            GlyphList.Cells[0, Index] := SearchRec.Name;//
Name
           GlyphList.Objects[1, Index] :=  Bitmap;  //
Bitmap
    inc(Index);
          except
         Bitmap.Free; { Executed if ANYTHING goes wrong }
           raise;      { Pass any exceptions up call
chain }
          end;
       end;
     K := FindNext(SearchRec);  { Continue directory scan
}
    end;
 finally
  FindClose(SearchRec);
   end;
finally
 Screen.Cursor := crDefault;  { Restore normal cursor }
  end;
end;

// Frees memory occupied by all glyph bitmaps. The
TStringGrid
// object does NOT do this automatically.
procedure TMainForm.FormClose(Sender: TObject;
```

```
    var Action: TCloseAction);
var
    I: Integer;
begin
    for I := 0 to GlyphList.RowCount - 1 do
      TBitmap(GlyphList.Objects[0, I]).Free;
end;

// Draw each glyph bitmap. The TStringGrid object is smart
// enough to draw its filename text objects with no further
// help. This code draws only the bitmaps in the second
column.
procedure TMainForm.GlyphListDrawCell(Sender: TObject;
    Col, Row: Integer; Rect: TRect; State: TGridDrawState);
var
    Bitmap: TBitmap;
begin
    if col = 1 then  // Be sure to refer to the bitmap column
    begin
        { Get bitmap object }
        Bitmap := TBitmap(GlyphList.Objects[1, Row]);
        if Bitmap <> nil then
        begin  { Draw bitmap in column cell }
      GlyphList.Canvas.BrushCopy(
              Bounds(Rect.Left + 2, Rect.Top + 2, Bitmap.Width,
                  Bitmap.Height), Bitmap,
                    Bounds(0, 0, Bitmap.Width, Bitmap.Height),
            clRed);
                { The preceding clRed argument gives the
        transparent
            glyph substance. Change this to any solid color,
          or
              change it to 0 to see why this is necessary. }
          end;
```

```
    end;
  end;

  end.
```

يعـــــرض الـ GlyphLst بعض التـقنيــــات الأسـاسـيـة التى يمكنك استخدامها فى جميع التطبيقات مع StringGrid object. تحتوى الـForm الرئيسية على ثلاث components فـــقط : Label1، BitBtn1، و GlyphList، من الـ TStringGrid class. (فى العـــادى كنت ســـأجـــعل اسم هذا الـ object GlyphStringGrid، ولكننى تركت الاسم GlyphList لأن النسـخة الأصلية من هذا البـرنامج تستـخـدم List object بدلاً من الـ component الأكـثـر تعـقيـداً (StringGrid).

لقـد قـمت بتـعـديل خـصـائص GlyphList object لإنشـاء قـائمـة ذات عـمودين. يعرض البرنامج اسماء ملف glyph فى العـمود الأول. يـوضح العمـود الثانى الـ bitmap المرتبطة بكل glyph. لبرمـجة الـ StringGrid الخاص بالتطبيق، قمت بتغيير هذه الخصائص الهامة :

- **ColCount = 2**: هذا هو عـدد الاعـمـدة والذى يمكن تغـييـره فى وقت التشغيل.

- **DefaultColWidth = 128**: يحـدد عـرض كل عـمـود. لان هـذه هى شبكة صفوف واعمدة، فكل خلية لها نفس العرض.

- **FixedCols = 0; FixedRows = 0**: هذه المواصـفـات تزيل رؤوس اعـمدة وصـفـوف الـ Grid التى تراها على الـ StringGrid الافتراضى. هذه القيم محـددة فى العـادى بـ 1 لتوفر عـموداً واحـداً وصـفاً واحـداً لمعلومات الرأس مثل الـ label الخاص للعمود.

قد تريد عادة ان تتولى رسم كل خلية فى الـ StringGrid . إنك تفعل هذا بإنشـاء الـ OnDrawCell الخاص بالـ StringGrid. وهذا يؤدى الى إنشـاء مـا هو مـعـــروف باسـم الـ owner drawn control. لكل خليـة فى الـ Grid، يقـــوم Delphi باستـدعـاء الـ event handler الذى يرسم مـا تريده داخل حدود الخلية.

وهناك خـاصـية أخـرى، وهى الـ DefaultDrawing ، تؤثر على كيفـية عـمل الـ StringGrid و الـ event handler الخاص بك معاً، كما هو موضح هنا:

- **DefaultDrawing = True**: ترسـم الـ StringGrid خلفـيــة الخليــة وتعرض ايه بيانات نص مرتبط بالخلية قبل استدعاءOnDrawCell . بعد إدخال الـ event handler، يرسم Delphi مستطيل حول الخلية التى تم اختيارها حالياً.
- **DefaultDrawing = False**: يعتبر OnDrawCell الخاص بك مسئول عن رسم كل خلفيات وواجهات كل خلية.

يســتــخــدم برنامج الـ GlyphLst الخـيــار الاول : =DefaultDrawing True . يملأ OnDrawCell الخـاص بالـ StringGrid كل زوج من الخــلايا باسم ملف الـ glyph's و bitmap المتعلق به . ان كل استـدعـاء GlyphListDrawCell procedure يحتوى على ثلاث معلومات هامة تم تمريرها كـ parameters .

- **Col**: رقم عمود الخلية.
- **Row**: رقم صف الخلية.
- **Rect**: حدود الرسم لمستطيل الخلية.

ان العمـود والصف الأولين كل منهـما يحمـل الرقم صفر. فى هـذه الحالة، لا يوجد لدينا قمم اعمدة أو صفوف- ولكن اذا استخدمتها، تذكر انك تحتاج ان ترسم هذه الخلايا ايضاً. ان GlyphList لا يستخدم الرؤوس.

هناك خـاصتين تحـملان الـ StringGrid's string وبيانات الـ object المرتبط به لكل خلية . يتم تعريف خاصية الـ Cells كما يلى :

```
property Cells[ACol, ARow: Integer]: string;
```

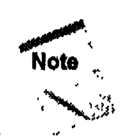
تعـتـبـر كل خليـــــة string object ، بسـيط وخـــــــال . يتم تعــريف خـاصـية الـ Objects باسلوب مشـابه، ولكن يمكنهـا ان تحـمل أى نوع من الـ class object، طالما انها تنحدر من الـ TObject:

```
property Objects [ACol, ARow: Integer]: TObject;
```

معاً، توفر خاصتى الـ Cells و الـ Objects الوصول الى الـ string والبيانات الاخرى لكل خلية من الـ StringGrid. ان عرض هذه البيانات هو مجرد توصيلها من خلال هذه arrays . لان بيانات النص يتم رسمها تلقائياً، يقوم الـ event handler الخاص بنا أولاً بالتأكد من ان الـ Col يساوى 1 . اذا كان كذلك، فإن الـ StringGrid يطلب ان يرسم التطبيق بيانات جرافيكية. وهذه لبيانات مخزنة فى خاصية الـ Objects للـ Grid. بفرض ان الـ Bitmap هو object مؤقت الـ TBitmap class، فهذه العبارة :

```
Bitmap := TBitmap(GlyphList.Objects[1, Row]);
```

تحصل على reference للـ Bitmap object المخزن فى العمود 1، والصف المشار إليه. بعد الحصول على هذا reference، يرسمه البرنامج باستدعاء الـ BrushCopy procedure الخاص بالـ StringGrid للـ Canvas التابع للـ object's . اختبر هذه العبارة فى القائمة، ولاحظ كيف يتم استخدام الـ Rect parameter لوضع الصورة داخل كل خلية. ان القيمة المتغيرة clRed فى هذه العبارة الـ glyph شفافاً.



ملحوظة: اذا كنت على دراية ببرمجة الـ Windows، فإن الـ Canvas يعادل الـ device context لا تهتم اذا لم تكن تعرف ما هى الـ device context سوف تعرف عنهم وعن الـ Canvas الكثير فى الباب ١٣ .

للإعداد لرسم كل Grid، يستخدم البرنامج OnCreate لنافذة الـ form الرئيسية. ولأن تحميل كل ملف glyph bitmap قد يستغرق وقتاً، يبدأ الـ FormCreate procedure بتحديد الـ Screen cursor بـ crHourGlass. ثم يستدع الـ FindFirst لوضع أول ملف مناسب. يتم إنشاء الـ Bitmap object بهذه العبارة.

```
Bitmap := TBitmap.Create;
```

ثم يتم استخدام هذا الـ object لتحميل bitmaps باستخدام العبارة :

```
Bitmap.LoadFromfile(glyphpath + '\' + SearchRec.Name;
```

وتقوم خاصية الـ Name فى متغير الـ SearchRec بالاحتفاظ بالنتائج من استدعاء الـ FindFirst . لتوفير مكاناً فى الـ Grid للاحتفاظ بـ bitmap واسم ملف آخرين، ينفذ البرنامج هذا الـ code :

```
if Index = GlyphList.RowCount then
  GlyphList.RowCount := Index + 1;
```

إن زيادة الـ RowCount بواحد يزيد كمية المعلومات التى يمكن ان تحملها وتعرضها الـ StringGrid. بعد ذلك تقوم عبارتان اضافيتان بتعيين bitmap واسم الملف لخلية الـ Grid :

```
GlyphList.Cells[0, Index] := SearchRec.Name;
GlyphList.Objects[1, Index] := Bitmap;
```

تحمل خاصية الـ Cells بيانات string كل خلية، والتى، فى هذا المثال، تم تعيين اسم ملف الـ glyph لها من سجل الـ SearchRec. يتم استخدام خاصية الـ Objects اختيارياً لربط بيانات اخرى بـ string كل خلية - فى هذه الحالة، يعين البرنامج glyph bitmap يكون قد تم تحميلها من القرص.

من المهم ان تفهم ان التعيين الى الـ Objects يربط Bitmap بهذه الخلية من الـ Grid. ولا تنسخ العبارة الـ Bitmap الى موضع جديدة لهذا السبب، يجب انشاء Bitmap object جديد باستدعاءTBitmap.Create لكل objectجديد تم تعيينه فى الـ Objects array فى الـ StringGrid.

تتواجد كل هذه الـ code داخل الـ try-except block، لذا اذا حدث أى خطأ- يحدث خطأ قرص، مثلاً، أو تنتهى ذاكرة البرنامج - يتم محو الـ Bitmap object الحالى ويتم تمرير الـ exception لمستدع الـ event handler . اذا سار كل شئ على ما يرام بعد معالجة كل glyph، يستدع البرنامج FindNext. استدع الـFindFirst لتبدأ العملية ثم استدع الـ .FindNext تكراراً حتى تدخل الـ function قيمة غير صفرية، مما يشير الى ان بحث اسماء الملف قد تم.

لاحظ ان الـ event handler يستخدم عبارتى finally فى نهايته، كلاً منها مرتبطة بـtry block. تنهى الـ FindClose بحث اسماء الملف، وتطلق Resources النظام الموجودة الـSearchRec object. من الصواب أن تستدع الـ FindClose داخل الـ finally block لضمان أن هذا الـ procedure تم استدعاءه حتى اذا حدث الـ exception فى مكان ما فى try block المرتبط به.

وهنـــاك فكرة صائبة أخـــرى، وإن كانت أقل أهمية، وهى أن تعيد تحديد الـ screen cursor بـ crDefault فى عـبـارة finally . هذا يضـمن أن cursor دخلت فى شكلها العادى حتى اذا حدث خطأ فى الـ procedure .

إن الـ StringGrid غير قادر على محو بيانات object تم تعيينها الى الـ Objects array . إنه يتحكم تلقائياً فى الذاكرة التى تستخدمها خاصية الـ Cells، ولكنها مسئوليتك أن تفعل هذا البنود مثل bitmaps المعنية للـ Objects . وهذا ضرورى لأن الـ StringGrid لا يعرف أى نوع من البيانات قمت بتعيينه للـ objects . بالرغم من أنه يتم استخدامه بشكل نموذجى للإشارة الى الـ Bitmap objects، فمن الممكن تعيين أى نوع من objects لهذا array . وهذا يأتى على حساب أن يجعلك مسئولاً عن تحرير أية ذاكرة مشغولة بهذا الـ objects .

والمكان الجيد لمسح الـ StringGrid object هو فى OnClose event handler الخاص بالـ form . فى الـ GlyphLst، مثلاً، محو الـ for loop التالية كل الـ Bitmap objects المخزنة كـ references فى الـ Objects array :

```
var
    I: Integer;
begin
    for I := 0 to GlyphList.RowCount - 1 do
      TBitmap(GlyphList.Objects[0, I]).Free;
end;
```

تشير خاصية الـ RowCount الى كم العناصر التى تم تخزينها فى الـ StringGrid . باستخدام هذه القيمة، فمن المستحسن إنشاء for loop تستدعى TBitmap.Free لكل Bitmap object مذكور لـ referenced من قبل الـ Objects array .

على القرص المدمج: لأغلب المعلومات عن قوائم الـ string، اختبر أيضاً التطبيق ToDo على القرص المدمج فى دليل الـ Source\Todo . يظهر هذا البرنامج تقنيات TStrings متنوعة. على سبيل المثال، توضح القائمة (٨-١١) الـ procedure الذى يستخدمه الـ ToDo لنقل عنصر مختار من احد الـ ListBox الى

آخـــر . لاحظ الـ TCustomListBox parameter . يمكنك اســتـــخـــدام الـ procedure Type TListBox's، ولكن اســتـــخـــدام الـ TCustomListBox يضـمن ان الـ procedure يعـمل لأية objects اخـرى من classes مـشـتـقـة من الـ TCustomListBox class .

القائمة (٨-١١): يستخدم ToDo هذا الـ procedure لنقل العناصر المختارة من ListBox الى آخر

```
procedure TMainForm.MoveSelected(List: TCustomListBox;
    Items: TStrings);
var
    I: Integer;
begin
    for I := List.Items.Count _ 1 downto 0 do
     if List.Selected[I] then
       begin
      Items.AddObject(List.Items[I], List.Items.Objects[I]);
            List.Items.Delete(I);
          end;
    FileDirty := True;
end;
```

يستدعى برنامج الـ ToDo الـ MoveSelected باستخدام عبارة مثل التالية، والتى تنقل المداخل المختارة بين الـ ListBox objects، من SrcList الى DstList :

```
MoveSelected(DstList, SrcList.Items);
```

فى الـ MoveSelected (راجع القـائمـة)، يعـتـبـر الـ parameter List هو الـListBox الخاص بالمسافة؛ والـ parameter Items هو الـ source TStrings . وتقـوم الـ for loop بقـيـاس الـ List من اسـفل الى اعلى (لاحظ اسـتـخـدم الكلمـة الرئيسية downto الخاصة بالـ Pascal). يجب ان يكون القياس فى هذا الاتجاه لان أى حذف فى البند N يغير فهارس البنود ....N+1.

اذا تم اخـتـيـار عنصـر قـائمة، ينسخ الـ AddObject الـ List's string وأى object مـرتبط به الى الـ parameter Items، من نوع الـTStrings . يحـذف

البرنامج بعد ذلك البند الافتراضى . ان استدعاء الـ Delete يحذف الـ string المشار إليـه والذى تم تمريره من خـلال الـ (I) index argument، وكـذلك يحـدد الـ Objects[I] بلا شئ . لذا، فإن استدعاء . واحداً للـDelete يزيل list's string وأى object مـرتبط به . ولكن، إنه لا يتـخلص من ذاكـرة هذا الـobject – إنك يجب ان تفعل هذا دائماً باستدعاء الـ Free .

افكار للمستخدم الخبير

● انظر مزيداً من المعلومات على رسائل _lb فى Windows API online الخاصة بـ Delphi. ان استدعاء الـ Perform لإسال رسالة الـ scroll bar الافقى فى list box مثلاً- يعتبر بشكل عام، ولكن ليس بالضبط، مساو لاستدعاء الـ Windows SendMessage function. يقوم الـ Perform باستدعاء الـ VCL WndProc ؛ يُستدعى الـ SendMessage للنافذة العلوية المرتبطة مع window handle.

● يمكنك ايضاً استخدام الـTStringList (أو خاصية الـ TStrings) لإنشاء strings lists و objects متعلقة. ان التقنيات تشبه تلك الموضحة فى تطبيق الـ GlyphLst، والتى تستخدم الـ StringGrid .

المشروعات التى يمكنك تجربتها

(٨-١) : اكتب تطبيقاً يقدم الـ ListBox لفرز ملف نص .

(٨-٢) : اكتب code تعكس ترتيب المداخلات فى الـ ComboBox object أو الـ ListBox.

(٨-٣) : اكتب procedure يحمل اسماء ملف دليل فى TStringList object .

(٨-٤) : قم بتحسين تطبيق الـ GlyphLst باضافة قائمة File ذات اوامر لتغيير الادلة. ملحوظة : لمنع البرنامج من محاولة تحميل صور

bitmaps ضخمة فى الـ ListBox، فقد تريد إضافة code ترفض الـ Bitmap objects الاكبر من الحجم المحدد سابقاً.

(٨-٥): متقدم. قم بإنشاء owner-draw ListBox يعرض string لها عدة أنماط من الـ Font. (ملحوظة: ان بعض الـ Fonts رمزية، ويجب ان تعرض اسم الـ Font منفصلاً عن الـ string الخاص به).

ملخص:

• ان الـ ListBox والـ ComboBox بـ Delphi لا تعتبر قيمة لإضافة قوائم قابلة للاختيار الى واجهات التطبيق للتطبيق. يمكنك الاختيار من بين عدد من الانماط لإنشاء قوائم متعددة الاختيارات والسماح للمستخدمين بادخال بيانات جديدة فى نافذة الـ ComboBox's edit.

• يقدم Delphi ايضاً classes صنع القوائم المرتبطة وهى TList، TStrings و TStringList. استخدم الـ TList والـ TStringList لإنشاء list objects مستقلة فى الذاكرة. لا يمكنك إنشاء TStrings objects -الـ components فقط يمكنها استخدم الـ TStrings كـ property's class type.

• ان كل الـ classes فى Delphi انحدر من الـ TObject. هذا يعنى ان كل class objects فى تطبيقاتDelphi مرتبطة.

• لان الـ TStringList تنحدر من الـ TStrings، فأى شئ يمكنك فعله بخاصية الـ TStrings (مثل خاصية الـ Items للـ ListBox). يمكنك ان تفعله مع TStringList مستقل.

• StringGrid يقدم الـ row-and-column general-purpose، والذى يمكنه ان يحمل string والمعلومات المتعلقة بكل خلية فى الـ Grid. بشكل نموذجى، تعتبر المعلومات المتعلقة بالـ Bitmap object، ولكنها يمكن ان تكون أى object آخر من class تنحدر من الـ TObject. ان خصائص الـObjects والـ Cells للـ StringGrid بـ arrays ثنائى الابعاد، والذى يمكن للبرنامج من خلاله ان يتوصل لمحتويات الـ Grid.

● يوفر الـ TStrings objects أو خاصية الـ TStrings الوصول الى بيانات الـ string من خلال خاصية Items ، أو من خلال الـ Items.Strings array الذى يساويه وتعتبر بيانات الـ object المرتبطة متاحة من خلال الـ Objects array .

● انها مسئوليتك ان تمحو الـ object ا التى تنشئها . وهذا يعتبر مهماً بشكل خاص عند استخدام الـ Objects array لـ StringGrid object . ان افضل مكان لمحو objects الخاصة بالـ Grid هو عادة فى OnClose الخاص بالـ form .

● تقوم الـ TList والـ TStringList تلقائياً بتحرير بيانات الـ string فقط . قمت بإضافة objects اخرى فى قائمة ، يجب ان تمحو تلك الـ objects باستدعاء الـ Free .

يمكنك ان تحدد عشرات الاستخدامات لـ string lists فى الـ component المؤسسة على نص والخاصة بـ Delphi مثل الـ Memo ، الـ Notebook ، والـ TabbedNotebook ، والتى تقابلها فى الباب التالى .



الباب التاسع
العمل مع Single-Line Strings

محتويات هذا الباب:

- Components
- Character strings
- Built-in text dialogs
- Single-line text components

فى العصر الماضى للحاسوب، كنا نحن المبرمجون محظوظون اذا كانت تطبيقاتنا بها حروف كبيرة وحروف صغيرة زائد القليل من رموز علامات الترقيم المعيارية. ولحسن الحظ، كان باستطاعتنا الاختيار من بين مجموعة كبيرة من الالوان- مادامت لا تخرج عن الاخضر والبرتقالى. اليوم، مع الـ Windows وواجهات تطبيق المستخدم الجرافيكية الأخرى، فإن الـ TrueType Fonts والشاشات كاملة الالوان تخلق فرص لا حدود لها لمصممى واجهات التطبيق، ولكن كاحدى النتائج، فقد واجه المبرمجون مجموعة جديدة تماماً من القواعد واللوائح للعمل مع الـ character strings و text objects أخرى.

يختبر هذا الباب single-line strings و text components من جانبين- التخزين الداخلى والعرض الخارجى. ان إنشاء واجهة تطبيق ناجحة لنظام يعمل بالجرافيك يتطلب الآن الانتباه الى العديد من تقنيات عرض وتخزين الـ strings الجديدة.

: Components

فيما يلى الـ Delphi's components للعمل مع الـ single-line strings :

• **Edit**: احد الـ controls ذات المظهر البسيط فى لـ Windows، ولكنه ايضاً واحداً من الاكثر تقلباً، فالـ Edit component يمكن ان يتعامل مع اغلب مهام single-line. يقوم الـ Edit تلقائياً بدعم عمليات القطع والنسخ واللصق، وله خصائص التعامل مع الفأرة ولوحة المفاتيح المركبة داخلياً. الـ Palette: Standard.

• **Label**: لقد استخدمت الـ Label component من قبل فى الابواب السابقة، ولكن الـ Labels لها استخدامات اخرى قد لا تكون واضحة. على سبيل المثال، يستطيع الـ Labels ان يحدد hot key لـ controls اخرى لا تؤيد اختيارات لوحة المفاتيح. الـ Palette: Standard.

• **MaskEdit**: قد تعتقد ان هذا الـ component هو Edit box ذا عقول. استخدم الـ MaskEdit لإنشاء حقول صالحة لإدخال بيانات- على سبيل المثال، مربع ادخال رقم الهاتف يتطلب عوامل لإدخال ارقام فى الاماكن المحددة. Palette: Additional.

Character Strings :

يقدم Delphi مجموعة من الـ string operators والـ functions والـ procedures. قبل إلقاء الضوء على الـ text components, والـ string، إنك تحتاج ان تفهم جيداً أثنان من الـ string formats بالـ Delphi وكذلك operators و functions الفصول التالية تقدم بعض المعلومات كخلفية عن نوع بيانات الـ string الخاص بـ Delphi.

Types of strings :

عندما انتقل Delphi 2 الى ساحة نظام تشغيل الـ ٣٢ بت، تم إجراء تحويل طويل لنوع بيانات الـ string الخاص بالـ Pascal القيمة. فى الماضى، كان الـ string عبارة عن array يصل الى ٢٥٦ رمزاً ذا ٨ بت، والبايت الأول فى هذا الـ array طول الـ string بالرموز. ويسمى بـ Length-Byte لا يزال متاحاً كـ ShortString فى الـ Object Pascal. والسبب الوحيد لاستخدام هذا الـ string حالياً هو للتلائم الخلفى.

والجديد فى الـ Object Pascal هو نوع بيانات للـ AnsiString . مع هذا النوع ، من الممكن ان توجد string ذات طول غير محدد .

فى اغلب الحالات ، تعتبر الـ string التى تقوم بانشاءها من هذا النوع . pascal's string data type تلقائياً يصبح من نوع هذا الـ String .

ملحوظة: ان الاحرف الصغيرة تشير الى أى نوع من أنواع الـ string فى حين ان بدء الكلمة بحرف كبيرة يشير الى الـ Object Pascal's data type بشكل خاص .

يمكنك تغيير نوع بيانات الـ string لتمثيل أى من نوعى الـ string باستخدام امر الـ $H compiler. فى الحالة الافتراضية ، {$H+} ، يعتبر الـ string هو نفسه الـ AnsiString . فى حالة الـ {$H-} ، يرجع الـ string الى طول البايت القديمة . من الافضل فى معظم الحالات ان تستخدم التعليق الافتراضى وان تستغل نوع بيانات الـ AnsiString الجديد . استخدم التحديد المقابل فقط اذا يجب عليك ان تجرى عملية الـcompile للـ code وتتوقع ان يكون الـ string هو طول البايت القديمة .

ونوع آخر من الـ string هو الـ WideString ، والذى يمكن ان يخزن رموزاً للغات تحتاج متغيرات ذات حروف أكبر . تقوم الـ WideString objects بتخزين الـ Unicode ، رموز ١٦ بت ، والتى تسمح بنص فى لغة معينة ان يكون اسهل نقلاً (وفى بعض الحالات يتم عرضه دون غموض) الى مجموعة من نظم الحاسوب .

ملحوظة: تستخدم الـ OLE ذات الـ ٣٢ بت الـ WideString لكل بيانات الـ string . وبالنسبة للتحكم فى الذاكرة ، يتم استخدام اثنان من الـ WideString objects والـ AnsiString بطريقة متشابهة ، كلاهما يمكن ان ينشىء متغيرات string ذات اطوال غير محددة .

String طويلة وقصيرة:

ان string يسهل إنشاءها واستخدامها . حدد متغير string باستخدام تعريف الـ var ، إما فى قطاع unit's implementation ، أو فى الـprocedure أو الـfunction :

```
var
    S: String;
```

هذا ينشئ string object خـال، والذى يمكن ان يحـمل من صـفـر الى أى عدد من الرمـوز. ويعرف الـ object أيضـاً dynamic string لان حـجـمه بتغـير ديناميكيـاً طبـقاً لطول بيـانات الـ string الفـعليـة التى تم تعـينهـا للـ object. بعـد تعريف متغير الـ string ، يمكنك تعيين string حرفى للـ S :

```
    S := 'Click!';
```

يمكن مـتغيـر الى S لخاصـيـة الـ Caption لـ control مـثل الـ Button. هذه العبارة، على سبيل المثال، تغير نص الـ Button object الى بيانات الرمز فى متغير الـ string :

```
    Button1.Caption := S;
```

يمكنك تنفيذ عمليات مشابهة لأى خاصية string. يمكنك ايضاً استخدام أى String object كـ array للرموز. ان الرمز الأول فى الـ string يحـمل الـ index 1، لذا فهذا التعيين :

```
    S[2] := 'X';
```

يعين X للرمز الثانى لمتغير الـ S string. الرموز الفردية هى من نوع بيانات الـ Char، وهى مـسـاوية للـ AnsiChar (رمـوز ذات ٨ بت). فى أغلب البـرامج، يمكنك استخدام متغيرات الـ Char و الـ string دون الاهتمام بانـواعها. ولكن، اذا كان يجب عليك استخدام رموزاً عريضة، يجب ان تعرف string objects الخاص بك من نوع الـ WideString ومـتـغـيـرات رمـوزك من نوع الـWideChar. ان Delphi لا يقـوم حـالياً بالانتقـال التلقـائى بين الـ ٨ بت العـادية و string الرمـوز العريضة.

يمكنك ان تستخدم string طول البايت الافتراضى الخاص بالـObject Pascal، اذا كنت تفضل هذا. يمكنك إنشاء واحداً من هذه الاشياء المعروفة الآن بـ ShortString، بطريقتين اساسيتين. استخدم اسم نوع البيانات هذا فى التعريف مثل :

```
var
    SS: ShortString;
```

هذا ينشئ متغير SS القادر على حمل ٢٥٥ رمزاً. ولان هذا عادة مضيعة للوقت (ان كل الـ strings القصيرة هى كلها ذات هذا الحد الأعلى من الطول)، فالطريقة الأخرى والأفضل عادة للإنشاء هى بتحديد حد أعلى للطول بين قوسين :

```
var
    SS: String[4];
```

لاحظ انك تستخدم تعريف الـ String، وليس الـ ShortString، لإنشاء ShortString object ذا حجم محدد. كما هو مقرر هنا، ان حاول الـ (SS) هو خمسة بايت- اربعة بايت لعدد رموز، وواحدة لطوله المخزن كقيمة البايت فى الـ SS[0].

Note

ملحوظة: يمكن ان تدفع الـ Short string الأداء سلباً- وهو سبب آخر جيد لعدم استخدامها. مثلاً، ان كل سبل استغلال الـ strings الخاصة بـ Delphi تقبل الـ strings parameters الطويلة. اذا قامت بتمرير strings قصيرة الى هذه السبل، فإنها تتغير داخلياً من والى الـ strings الطويلة، مما يهدر الوقت والذاكرة. لأفضل أداء، استخدم نوع بيانات الـ string الخاصة بالـ Pascal، واستخدم الـ ShortStrings فقط عند الحاجة الماسة.

يمكنك مقارنة الـ String objects ابجدياً باستخدام العوامل المنطقية المعتادة: > ، < ، => ، <= ، >= ، و= . على سبيل المثال، الـ code التالى توضح كيفية مقارنة إثنين من الـ String objects ابجدياً:

```
var
    S1, S2: String;
begin
    S1 := 'ABCDEFG';
    S2 := 'abcdefg';
    if (S1 > S2) then
        ShowMessage('S1 > S2')
    else
     ShowMessage('S1 <= S2');
end;
```

ملحوظة لمبرمجى الـ C والـ C++ : من الممكن ايضاً استخدام الـ Object Pascal لتحديد null-terminated string كـ array للـ Char . هذا ينشئ نوع

بيانات مســــاو لـ null-terminated string في برمـجـة لغـات مـثل الـ ++C .
على سبيــــل المثال، ما يلى يحدد string من ٦٤ رمزاً زائد بايت واحد يحتفظ به للـ terminating null :

```
var
    S: array[0 .. 64] of Char;
```

بالرغم من ان هذا مسموح به، الا انه من افضل استخدام الـ strings الخاصة بالـ Pascal بدلا من الـ null-terminated character arrays . ان الـ String function بالـ Delphi فعالة للغاية، اسهل في الاستخدام، ويمكن الاعتماد عليها أكثر من كتابة functions خاص بك- حتى اذا كنت على دراية بالـ C والـ ++C والـ null-terminated strings والـ string pointers . كقاعدة عامة، إنك تحتاج الـ null-terminated strings لتمرير pointers الى الـ Windows functions أو للإستـخدام مع functions و library procedures التى تتطلب هذا النوع من البيانات.

ملحـوظة: ان الـ strings الطويلة تكون دائمـاً الـ null-terminated strings بالرغـم من أنه لايوجـد function أو procedure يتطلب هذا كـقـاعـدة. ولأنهـا null-terminated، فـالـstrings الطويلة تعـد type-cast متماشياً مع الـ PChar pointer .

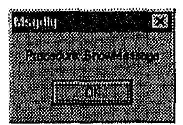
: Built-In Text Dialogs

اذا لـــم تكن قـــد إكتشفت string input and output dialogs وفى الـ Delphi، فإننى اذكرها هنا إنها ادوات سهلة الاستخدام لعرض رسائل وحث المستـخـدمين على الادخـال جـرب هذه- ان text single-line utilities لهـا عشرات الاستخدمات.

: Displaying text messages

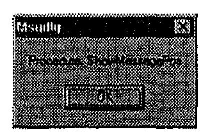
يقدم Delphi خمسة message dialogs، واحدة منها من المؤكد ان تفى باحتياجاتك. توضح الأشكال من (٩-١) الى (٩-٥) الـdialogs الخمس بترتيب درجة التعقيد.

● يعرض الـ ShowMessage procedure الرسائل الثابتة فقط، كما هو موضح في شكل (٩-١).



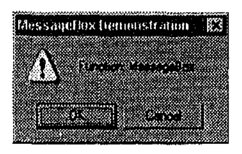
شكل (٩-١): يعرض الـ ShowMessage
الـ string في dialog box مع زر OK

● ان الـ ShowMessagePos procedure هو نفسه الـ ShowMessage ولكن باحاثيات الـ -x و الـ -y التي تحدد موضع dialog box بالنسبة للشاشة [انظر شكل (٩-٢)].



شكل (٩-٢): تحتوى الـ ShowMessagePos dialogs
على احداثيات -x و -y

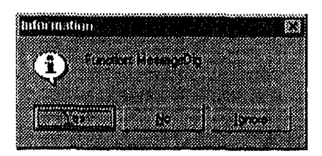
● يقـــوم MessageBox procedure باحـــتـــواء الـ Windows API MessageBox function. في شكل (٩-٣) في الـ Windows 3.1، تعرض الـ functions، displays غيـر الـ 3D، قد يتم الاسلوب. للحصـول على نتـائج افضل، استخدم واحدة من functions التالية بدلاً منه. في الـ Windows 95، ينتج الـ MessageBox نفس النتـائج البـصـرية كـالـ MessageDlg والـ MessageDlgPos. (جـمـيـع الـ controls في الـ Windows 95، NT، والـ Windows 98 لها مظهر 3D).



شكل (٩-٣): الـ MessageBox dialog يحتوى
الـ Windows API MessageBox function

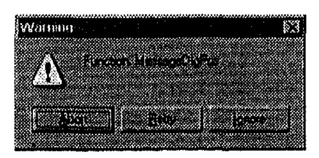


● يقدم الـ MessageDlg procedure مجموعة واسعة من الخيارات لعرض الرسائل والحث على إجابة الاسئلة . يمكنك اختيار ايقونة [مثلاً ، دائرة المعلومات الموضحة فى شكل (٩-٤)]، ويمكنك اختيار انواع وعدد من الازرار التى تقدمها .



شكل (٩-٤): ان الـ MessageDlg dialog box يمكنك من اختيار ايقونة، مثل دائرة المعلومات الموضحة هنا، للعرض

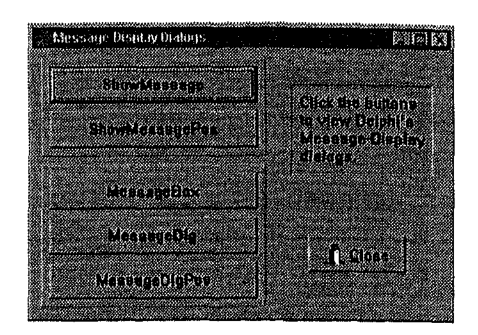
● ان الـ MessageDlgPos procedure هو نفسه الـ MessageDlg بخلاف انه يقدم إحداثيات -x و -y لوضع نافذة الـ dialog فى أى مكان على شاشة الحاسب المكتبى للـ Windows، كما هو موضح فى شكل (٩-٥) .



شكل (٩-٥): يمكنك وضع الـ MessageDlgPos dialog فى أى مكان حاسب المكتبى للـ Windows

الـMessageDlg والـ MessageDlgPos عبارة عن procedures تعرض رسائل . وهى مناسبة فقط لعرض الرسائل الثابتة . الـ MessageBox، والـ MessageDlg، و الـ MessageDlgPos تعتبر functions، وهى تدخل قيمة تشير الى أى من الازرار تم ضغطه لإغلاق نافذة الـ dialog استخدم واحدة من هذه الـfunctions لحث المستخدمين على إجابة اسئلة أو تاكيد اختيار- مثلاً، " هل انت متأكد تماماً من انك تريد مسح محرك قرصك الصلب؟ " .

يوضح تطبيق الـ MsgDlg على القرص المدمج فى دليل الـ Source\MsgDlg الانواع الخمسة من الـ dialogs لعرض الرسالة . ان جميع الطرق الا واحدة يسهل استخدامها . أما الـ MessageBox فتطلب استخدام تقنيات string لم يتم تقديمها بعد . سوف اشرح هذا عقب القائمة . يوضح شكل (٩-٦) عرض البرنامج . تظهر القائمة (٩-١) الـ source code .



شكل (٩-٦): قم بتشغيل الـ MsgDlg لتجربة
الانواع الخمسة للـ dialogs لعرض الرسالة

القائمة (٩-١): Msgdlg\Main.pas

```
unit Main;

interface

uses
  SysUtils, Windows, Messages, Classes, Graphics,
    Controls, Forms, Dialogs, StdCtrls, ExtCtrls, Buttons;

type
    TMainForm = class(TForm)
      Button1: TButton;
        Button2: TButton;
        Button3: TButton;
        Button4: TButton;
        Button5: TButton;
        BitBtn1: TBitBtn;
        Label1: TLabel;
        Panel1: TPanel;
        Bevel1: TBevel;
        Panel2: TPanel;
        procedure Button1Click(Sender: TObject);
       procedure Button2Click(Sender: TObject);
```

```
    procedure Button3Click(Sender: TObject);
      procedure Button4Click(Sender: TObject);
      procedure Button5Click(Sender: TObject);
    private
   { Private declarations }
    public
   { Public declarations }
    end;

var
   MainForm: TMainForm;

implementation

{$R *.DFM}

procedure TMainForm.Button1Click(Sender: TObject);
begin
 ShowMessage('Procedure: ShowMessage');
end;

procedure TMainForm.Button2Click(Sender: TObject);
begin
   ShowMessagePos('Procedure: ShowMessagePos', 10, 20);
end;

procedure TMainForm.Button3Click(Sender: TObject);
var
 TheText, TheCaption: String;
begin
 TheText := 'Function: MessageBox';
   TheCaption := 'MessageBox Demonstration';
   if Application.MessageBox(PChar(TheText), PChar
  (TheCaption),
```

```
      MB_DEFBUTTON1 + MB_ICONEXCLAMATION + MB_OKCANCEL) = IDOK
        then ShowMessage('You selected OK')
        else ShowMessage('You selected Cancel');
end;

procedure TMainForm.Button4Click(Sender: TObject);
var
   W: Word;
   S: String;
begin
   W := MessageDlg('Function: MessageDlg',
    mtInformation, [mbYes, mbNo, mbIgnore], 0);
     case W of
    mrYes:   S := 'Yes';
    mrNo:    S := 'No';
    mrIgnore: S := 'Ignore';
   end;
   ShowMessage('You selected ' + S);
end;

procedure TMainForm.Button5Click(Sender: TObject);
var
 W: Word;
   S: String;
   X, Y: Integer;
begin
   X := 50; Y := 75;
   W := MessageDlgPos('Function: MessageDlgPos',
    mtWarning, mbAbortRetryIgnore, 0, X, Y);
     case W of
    mrAbort:  S := 'Abort';
      mrRetry:  S := 'Retry';
      mrIgnore: S := 'Ignore';
```

```
    end;
      ShowMessage('You selected ' + S);
  end;

  end.
```

ان الـ ShowMessage سـهلة الاسـتـخـدام. فـقط قـم بتـمـرير ايه string أو string variable لعـرض رسـالة تذكـر ان Delphi يسـمـح لك بربـط الـ strings بعـامل الزائد- وهذا مـا يجـعل الـ ShowMessage سـهلة لعـرض كل انواع القـيم ولإزالة اخطاء المتغـيـرات. على سـبيل المثال، لعـرض قـيمـة مـتـغـيـر الـ Integer Count، استخدم عبارة مثل هذه:

```
    ShowMessage('Count = ' + IntToStr(Count));
```

والـ ShowMessagePos هو نفسه الـ ShowMessage ولكن يضـيف قيم احداثيات الـ -x و الـ -y، ذات عـلاقة بالشـاشة. لعرض الرسالة السـابقة فى الركن الايسر العلوى لحاسب الـ Windows المكتبى، استخدم هذا العبارة:

```
    ShowMessagePos('Count = ' + IntToStr(Count), 0, 0);
```

يدخل الـ MessageDlg والـ MessageDlgPos قيمة تشير الى الزر الذى تم ضـغطه لإغـلاق نافـذة الـ dialog. لانك تقـرر أى الازرار يـعـرضـه الـ dialog، فعليك ان تبحث عن هذه الازرار فقط- إنك لا تحتاج الى كتابة كود للاستجابة لكل قيـمة مـدخلة محتملة. ان عـبارة case عادة تكون افضل إنشـاء أولاً، حـدد متغير الـ Word وهو W وعين له MessageDlg أو MessageDlgPos بعبارة مثل هذه:

```
  W := MessageDlg('Function: MessageDlg',
    mtInformation, [mbYes, mbNo, mbIgnore], 0);
```

ثم استخدم عبارة case للقيام بعـمل مناسب يعتمد على الزر الذى اختاره المستخدم:

```
  case W of
      mrYes:    { action for Yes    };
      mrNo:     { action for No     };
      mrIgnore: { action for Ignore };
  end;
```

ان الـ MessageDlgPos هى تقسـهـا الـ MessageDlg ولكن تضـيف احداثيات الـ -x و الـ -y ، لوضع الـ dialog على حاسب الـ Windows المكتبى . ان الاحداثيات لها علاقة بالشاشة ، ولكن يمكنك تحويلها الى قيم مرتبطة بالـ client لعرض dialogs داخل مساحة نافذة البرنامج . وهذا افضل لواجهة التطبيق لان المستخدمين يميلون الى البحث عن الرسائل فى مواضع محددة مرتبطة بنافذة التطبيق .

ان الـ MessageBox يعتبر اكثر تعقيداً فى الاستخدام . إنك تحتاج ان تمرر له null-terminated strings زائد مجموعة منطقية من الثوابت مثل الـ MB_OKCANCEL لاختيار خيارات متنوعة :

```
function MessageBox(Text, Caption: PChar;
  Flags: Longint): Integer;
```

الأثنان من parameters من نوع الـ PChar ، والتى تمثل pointer الى رمز فى Delphi . لان الـ strings الطويلة تكون null terminated ، يمكنك وضعهم فى الـ PChar pointers للتمرير الى الـ Windows functions مثل هذه . على سبيل المثال ، يقرر الـ Button3Click declare إثنين من متغيرات الـ string :

```
var
    TheText, TheCaption: String;
```

يقوم البرنامج بتعيين strings حرفية الى هذه المتغيرات باستخدام عبارتين :

```
TheText := 'Function: MessageBox';
TheCaption := 'MessageBox Demonstration';
```

قم بتمرير أثنان من الـ string ، type-cast إلى الـ PChar pointers ، من ثم إلى الـ MessageBox function . يقوم البرنامج بهذا باستخدام عبارة if لاختيار نتيجة الـ function ، والتى تشير الى الزر الذى تم ضغطه لإغلاق نافذة dialog box توضح عبارة الـ if كيف تضع strings فى الـ PChar pointers :

```
If Application.MessageBox(PChar(TheText), PChar
  (TheCaption),
    MB_DEFBUTTON1 + MB_ICONEXCLAMATION +
  MB_OKCANCEL) = IDOK
    then ShowMessage('You selected OK')
    else ShowMessage('You selected Cancel');
```

ان التعبيرات (الـ PChar (Text و(الـ PChar (Caption تمــرر متغيرات الـ string الطويـل الى الـ Windows function ، والتـى تتــــوقـع أن تتلقى null-terminated strings فى هذه المواضع . تقوم الـ MessageBox function بادخال قيمة الزر الذى ضغطه المستخدم . اذا كانت مساوية لـ IDOK، اذن يتم إظهار الرسالة الأولى باستخدام الـ ShowMessage . اذا لم تكن كذلك، يكون المستخدم قد ضغط Cancel، ويتم عرض الرسالة الثانية .

وتوضح الـ Button3Click فى القائمة كيفية استخدام الـ MessageBox، ولكننى اقــتــرح عليك أن تســتــخــدم بدلاً منه الـ MessageDlgPos والـ MessageDlg اللذان يتشـابهـان فى التـصـمـيم ويقـبـلان الـ Pascal string parameters . يقـوم Delphi بتـعـريف الـ MessageDlg والـ parameters الخاصة به كما يلى :

```
function MessageDlg(const Msg: string; AType: TMsgDlgType;
  AButtons: TMsgDlgButtons; HelpCtx: Longint): Word;
```

- **Msg:** تمرر string ثابت أو متغير حرفى للعرض فى dialog box .

- **AType:** حــدد هذا الـ parameter بثـابت مــثل الـ mtWarning أو الـ mtError لإختيار الأيقونة المناسبة و dialog caption .

- **AButtons:** حدد هذا الـ parameter بمجموعة الأزرار التى تريد عرضها فى الـ dialog . ضع المجمـوعـة بين قـوسين . على سبيل المثال، مـرر التعـبـير [mbYes, mbNo, mbCancel] لعرض أزرار Yes أو No أو Cancel . يضيف Delphi تلقـائيـاً الـ glyphs المناسـبـة لكل زر (إنهـا الـ BitBtn component object) . وبالعكس، بدلاً من تحديد مجموعة الأزرار الخاصة بك، قم بتمرير مجموعة سابقة التحديد، مثل الـ mbYesNoCancel، mbOkCancel أو myAbortRetryIgnore . فهذه هى بالفعل مجموعات Pascal- لا تحيطها بالأقراص .

- **HelpCtx:** حدد هذا الـ parameter بصفر الا اذا كنت تحدد نص مساعدة message box .

انظر الـ MessageDlg لمزيداً من المعلومات عن القيم الممكن ان تمررها لهذه الـ function وللـ MessageDlgPos .

On the CD-ROM

على القرص المدمج: توضح القائمة (٩-٢) كيفية استخدام الاحداثيات المتعلقة بالشاشة لوضع عناصر فى النافذة. حدد متغير الـ TPoint بـ P ، حدد اعضاء الـ X والـ Y التابعين له بـ (0)، واستدع الـ ClientToScreen، والذى يدخل سجل TPoint بهذه القيم تم تحويله الى إحداثيات شاشة مساوية. يمكنك عندئذ استدعاء MessageDlgPos كما هو موضح (أو استدع ShowMessagePos) لوضع message box فى الركن الايسر العلوى أو فى موضع آخر من الـ client area بنافذة التطبيق. يوجد نص القائمة على القرص المدمج فى دليل Strings فى ملف الـ Position.pas .

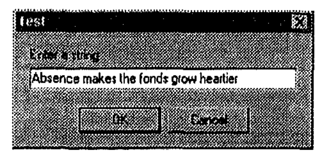
القائمة (٩-٢): استخدم هذا الـ code لوضع message dialog داخل نافذة التطبيق.

```
var
    W: Word;    { Result of MessageDlgPos }
    P: TPoint;  { Coordinate X, Y record }
begin
    P.X := 0;  { Assign client X value }
    P.Y := 0;  { Assign client Y value }
    P := ClientToScreen(P);  { Convert P to screen coordinates
   }
    W := MessageDlgPos('Function: MessageDlg',
      mtInformation, [mbYes, mbNo, mbIgnore], 0,
         P.X, P.Y);  { Pass converted X, Y to function }
end;
```

اذا قمت بتمرير الـ X والـ Y مساوية لـ 1- للـ ShowMessagePos أو الـ MessageDlgPos، تعرض الـ functions نوافذ الـ dialog الخاصة بها فى الاماكن الافتراضية التى اختارها الـ Windows .

الحث على إدخال النص:

على القرص المدمج: ان البرمجة عادة هى عملية اخذ وعطاء. ان الـmessage dialogs السابقة تعطى المستخدمين معلومـــــات. لكى تأخذ إدخــــالاً من المستخدمين، استدع الـ InputBox functions أو الـ InputQuery والتى تعـــــرض الـ dialog الموضح فى شكل (٩-٧). الـ InputBox ترجع بـ string؛ الـ InputQuery ترجع بقيمة Boolean بـ True أو False ، ولكن فيما عدا هذا تعتبر كل الـ functions متساويتين. توضح القائمة (٩-٣) كيفية استخدام الـ InputBox والـ InputQuery . انسخ هذا الـ procedures الى وحدة برنامج مثل OnClick لإثنين من الـ Button objects على الـ form، وقم بتشغيل البرنامج لتجربته مع الـ InputBox functions والـ InputQuery. هذا ليس برنامجاً كاملاً- لتستخدم الـ code، أضف اثنان من الـ Button objects علـى الـ form وانسـخ الـ procedures فـى OnClick لكل Button. يوجد نص القائمة على القرص المدمج فى دليل Strings فى ملف Input.pas.



شكل (٩-٧): استدع الـ InputBox functions أو الـ InputQuery لحث المستخدمين على إدخال single-line text strings تعرض الـ functions الـ dialog الموضح هنا

القائمة (٩-٣): استخدام الـ InputBox والـ InputQuery

```
 procedure TForm1. Button1Click (Sender: Tobject);
var
     S: String;
begin
  S := InputBox('Test', 'Enter a string', '');
     if Length(S) > 0 then
          ShowMessage('You entered: ' + S);
end;
```

```
procedure TForm1.Button2Click(Sender: TObject);
var
    S: String;
begin
   S:='' ;
   if InputQuery('Test', 'Type QUIT to end', S) then
     if Uppercase(S) = 'QUIT' then
         Close;
end;
```

تأخذ الـ InputBox ثلاث string guments وهم الـ caption، وprompt، وقيمة string ، والتي يمكن ان تكون متغيرة أو ثابتة. تمرر القائمة null string للـ InputBox للـ string الإفتراضي، ويتم إدخاله اذا لم يدخل المستخدمون اية تغييرات. يعرض الـ InputBox الـ string الإفتراضي وقد تم إبرازه فى الـ dialog وتقوم الـ function ايضاً بادخال الـ string الإفتراضى اذا اضغط المستخدم زرالـ control. اذا اضغط المستخدم زر الـ OK، الـ InputBox يرجع النص الذى تم إدخاله فى الـ Edit contro للـ dialog. على سبيل المثال، استخدم الـ InputBox للحث على إدخال اسم ملف :

```
S := InputBox('Test', 'Enter filename', 'Readme.Txt');
ShowMessage('You entered: ' + S);
```

On the CD-ROM **على القرص المدمج:** ان الـ InputQuery هو نفسه الـ InputBox، ولكنه يدخل True اذا ضغط المستخدم زر الـ OK، أو False اذا ضغط زر الـ Cancel. يضع الـ InputQuery إدخال المستخدم فى الـ string الإفتراضى ، والذى يجب ان يكون متغير. تعتبر الـ functions نموذجية فى حث المستخدمين على إدخال الـ strings مثل اسماء الملفات الإفتراضية للمسارات المعروفة. على سبيل المثال، أضف Button و Memo على الـ form ، ثم استخدم البرمجة الموجودة فى القائمة (٩-٤) للحث على ادخال اسم ملف، وهو الافتراضى الـ Readme.txt الخاص بهذا للكتاب فى دليل الـ Source. يتم توضح نص الملف المختار فى Memo object. يوجد نص القائمة فى القرص المدمج فى دليل Strings فى ملف Query.pas. .

القائمة (٩-٤): كيفية تحميل ملف معين

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Filename: String;
begin
  Filename := '\Delphi 4 Bible\Source\Readme.Txt';
    if InputQuery('Test', 'File?', Filename) then
      Memo1.Lines.LoadFromFile(Filename);
end;
```

Single-Line Text Components :

يقـدم Delphi ثلاث components إخـراج وادخـال نص احـادى السـطر، وهى: Label، Edit، و MaskEdit . أنهـا components سـهلة الاسـتـخـدام نسبياً، ولكن الفصول التالية توضح التقنيات الغامضة التى يجب ان تعرفها .

Labels :

ان الـ Labels ببسـاطة text objects يمكنك إضـافـتـهـا على forms و components أحادية اخرى، مثل الـ Panel . كما اكتشفت من خلال امثلة هذا الكتاب حتى الآن، ان للـLabels خصائص متعددة . يمكنك تحديد الـ Font الخاص بالـ Labels لتـغـيـيـر نمط نصـه، يمكنك عـرض wrap-around بتـحـديد الـ WordWrap بـ True، يمكنك تغـيـير لون الـ Labels من خـلال خـاصـيـة الـ Color .

Tip **فكرة:** تختار الـ Color لون خلفيةالـ Labels . استخدم خاصية الـ Font لاختيار لون نص واجهةالـ Labels .

فى اغلب الحالات، تعتبرالـ Labels غير متفاعلة- فهى ببساطة تضع label لـ component آخر، أو تعطى بعض معلومات على الشاشة . وتعتبر واحدة من الاستخدمات الغير واضحة للـ Label هى التعامل معه كـ accelerator hot key لـ control آخـر لا يكون فى العـادة مـتـوافق مع اخـتـيـارك لأخـذ عناصـر الـ keyboard . إتبع هذه الخطوات لتجرب هذه التقنية المفيدة :

١- أضف ثلاثة Edit objects و ثلاثة Label objects على form . ضع label الى اليسار مباشرة من كل الـ Edit control .

٢- غير الـ Caption الخاص بالـ Label1 الى &One . تحدد علامة الـ & حرفاً تالياً (وهو O فى هذه الحالة) كـ accelerator hot key .

٣- اختر خاصية الـ FocusControl للـ Label1، اضغط السهم المشير الى أسفل، واختر Edit1. عندما يتلقى الـ Label1 input focus، فإنه يمرر الـ focus الى الـ control المشار إليه .

٤- أعد الخطوات ٢ , ٣ للـ Label الآخرين. حدد الـ Caption الخاص بالـ Label2 بـ &Two، والخاص بالـ Label3 بـ T&hree . عين الـ FocusControl الخاص بالـ Label2 للـ Edit2، والخاص بالـ Label3 للـ Edit3 .

٥- قم بتشغيل البرنامج واضغط Alt+O، Alt+T و Alt+H لتحويل الـ focus الى الـ labeled Edit boxes .

لوضع الـ Label أفقياً مع الـ Edit أو أى control آخر متعلق به، اختر أولاً الـ Edit control، اضغط واستمر فى ضغط مفتاح الـ Shift، واضغط الـ Label المقابل. اختر أمر الـ ...EditlAlign الخاص بـ Delphi، اختر Vertical Centers radio button، واضغط Enter أو OK .

الـEdit boxes:

للتعامل مع single-line input، لا يوجد ما يضاهى الـ Edit component الخاص بـ Delphi . فيما يلى بعض الأفكار عن الخصائص المختارة الـ Edit component .

* حدد الـ AutoSelect بـ True لإبراز محتويات الـ Edit عندما يتلقى الـ input focus .

* حدد الـ CharCase بـ ecNormal للإدخال العادى حدده بـ ecUpperCase أو ecLowerCase لقصر الأدخال على الأحرف الكبيرة أو الحروف الصغيرة .

* حـدد الـ HideSelection بـ False اذا كنت تريد نص تم إبرازه أن يبـقى مـخـتـاراً عندمـا يحـول المسـتـخـدمـون الـ focus الى control آخـر . حـدد الـ HideSelection بـ True العملية العادية- النص المختار غير محدد عندما ينتقل الـ focus من الـ Edit control .

* حدد الـ MaxLength بأكبر عدد من الرموز تريد أن يدخلها المستخدمون . بالرغم من أنه بإمكانك أن تحدد هذه الخاصية بصفر للحصول على طول للادخال غير محدد ، فمن الناحية العملية ، إنك تستخدم قيم صغيرة نسبياً . إن حجم النص الذى يمكنك إدخاله فى Edit boxes هو فى الحقيقة غير محدد . فى النسخ الأولى لـ Delphi ذات الـ ١٦ بت ، كان النص محدد بـ ٢٥٥ رمزاً . ولقد تم إزالة هذا الحد فى نسخ الـ ٣٢ بت من Delphi؛ ولكن الـ Windows 95 و 98 (ولكن ليس الـ NT) يقصرا النص على 64K كحد أقصى .

* إن الـ Edit component ليس له خاصية Caption . استخدم Label ، كـمـا هو مـوضـح فى الفـصل السـابق ، لإعطاء الـ Edit objects الـ captions و hot-key accelerators لعرض object ، أضف رموزاً فى خـاصـيـة الـ Text ، أو عين string فى وقت التشغيل .

* لعـرض رمـوز DOS على نص الشـاشـة ، حـدد الـ OEMConvert بـ True ، وغـيـر الـ Font بـ MS LineDraw TrueType . حـدد الـ OEMConvert بـ False اذا لم تكن تريد رموز ANSI محـولة (هذا هـو التـحديد الطبيعى لـ Windows) . لتأييد رموز الـ ASCII الممتدة فى اسم ملف الـ DOS ، يجب أن يكون الـ OEMConvert محدد بـ True .

وتوفر الـ Edit components أيضاً events عديدة . وبعض هذه الـ events المفيدة هى :

• **OnChange:** يتم استدعاءه عندما تتغير محتويات الـ control بأية طريقة . اذا قمت بتشغيل هذا الـ event ، فإنتبه الى أنه يتم استدعاءه لكل رمز يتم إدخاله فى الـ control . للتعامل مع الحروف حرفاً حرفاً والمدخلة فى الـ Edit control ، فيجب أن تستخدم الـ OnKeyPress event .

• **OnClick:** يتم استدعاءه عندما يضغط المستخدم الفأرة مرة واحدة في الـ Edit control. لاحظ أن الـ event handler هذا لا يتم استدعاءه عندما يتلقى الـ Edit control الـ focus بطرق أخرى- عندما يضغط المستخدم الـ Tab مثلاً.

• **OnDblClick:** يتم استدعاءه عندما يضغط المستخدم الفأرة مرتين في الـ Edit control. والإستجابة العادية لهذا الـ events تنسخ خاصية الـ Text لـ Edit control الى object آخر مثل الـ ListBox.

• **OnEnter:** يتم استدعاءه عندما يتلقى الـ control الـ input focus- أي عندما يختار المستخدم الـ control ويضغطه بالفأرة.

• **OnExit:** يتم استدعاءه عندما يفقد الـ control الـ input focus- أي عندما يحول المستخدم الـ focus الى control أخر.

• **OnKeyPress:** يتم استدعاءه مع كل ضغطه مفتاح. بشكل عام هذا الـ event هو الذى يصح استخدامه لتنقية إدخال لوحة المفاتيح على سبيل المثال، لمنع المستخدمين من ضغط مفاتيح معينة، اذا كان الـ Key parameter الخاص بالـ event handler يساوى قيمة مستهدفة، حدد الـ Key بـ #0 حتى تبطله.

استخدام الـ OnExit والـ OnEnter:

من الناحية العملية، تعتبر الـ OnEnter والـ OnExit مفيدة في تصميم واجهات تطبيق جذابة. على سبيل المثال، يمكنك استخدام هذه الـ events لتلوين الـ focus Edit control، والذى يمكن أن يساعد في لفت إنتباه المستخدم الى المجال الحالى في نافذة ذى مساحات إدخال متعددة. عرف متغير OldColor من نوع الـ TColor في القطاع private في الـ form class، كما يلى:

```
TForm1 = class(TForm)
  Edit1: TEdit;
private
  OldColor: TColor;
...
public
...
end;
```

بعد ذلك، قم بإنشاء OnEnter للـ Edit. عين الـ Color الخاص بالـ object لمتغير الـ OldColor ثم اختر اللون الذى تريده لخلفية الـ control. مع هذا الـ code، عندما يقوم المستخدمون بتحويل الـ focus الى الـ control، كيف تتغير خلفيته الى اللون الأصفر:

```
procedure TForm1.Edit1Enter(Sender: TObject);
begin
    OldColor := Edit1.Color;
    Edit1.Color := clYellow;
end;
```

وكـذلك، قم بالـ event handler للـ OnExit event الخـاص بالـ Edit object. عين قيمة الـ OldColor لخاصية الـ Color للـ object. عندما يقوم المستخدمون بتحويل الـ focus عن الـ control، يستعيد البرنامج خلفية الى اللون الذى تم حفظه.

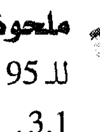
```
procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := OldColor;
end;
```

:Telephonic mnemonics

على القرص المدمج: إن تطبيق الـ TelName، الموجود على القرص المدمج فى دليل الـ Source\TelName، يظهر كيفية استخدام الـ Edit objects. وهذا البرنامج يستخدم Delphi لرسم وجه جديد على بعض الـ codes التى كتبتها لعمود Dr. Dobb الخاص بى، وهو "Algorithm Alley" منذ عدة سنوات. ولقد حل البرنامج واحدة من مشاكلى الدائمة- سرعة التذكر للصيغة الهاتفية الموجودة فى اعلانات مثل "اتصلوا بنا الآن، فقط اتصل بـ ١-٨٠٠- اتصل- بنا- الآن". وهذا يستغرق منى وقتاً طويلاً لأجد الأرقام المقابلة فى قرص الهاتف، لذا كتبت برنامجاً لتحويل الصيغة الى أرقاماً. ولقد كتبت أيضاً الـ code العكسية التى تعرض كل البدائل الممكنة لأى رقم. فى منطقتى مثلاً، الإستبدال العام ٢٩٣ يمكن أن ينهى كلمات مثل AWE، AXE، و AYE، مؤدياً الى صيغ هاتفية ممتعة.

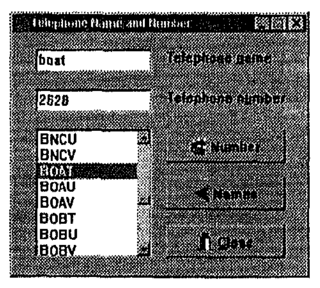
لاستخدام الـ TelName، أدخل صيغة هاتفية في Edit control لاسم Telephone. اضغط زر الـ Number لتحويل الاسم الى ما يوازيه بالأرقام. أو، أدخل رقماً في الـ Edit control لاسم الـ Telephone، واضغط Names لإدخال كل بدائل الصيغ الممكنة في list box النافذة. ملحوظة: على قرص الهاتف، الأرقام واحد وصفر ليس لها حروف مقابلة، والحروف Q و Z ليس لها أرقام مقابلة.

إن أرقام الهاتف الطويلة في الـ TelName تنتج مداخل ListBox متعددة، والتي قد تؤدي الى بالـ Windows الى حالة run out of resources. إن عدد المدخلات يساوي 3n و n هي عدد الأرقام. فالأربع أرقام تنتج ٨١ مدخلاً. والستة أرقام تنتج ٧٢٩ مدخلاً. للحصول على أفضل النتائج، اكتب ثلاثة أو أربعة أرقام كحد أقصى في الـ Edit لرقم الـ Telephone قبل ضغط الـ Names. على سبيل المثال، لترى ما عناصر رقم هاتفك، بدلاً من إدخال كل الأرقام، أدخل code المنطقة، والبديل له رقم بصورة منفصلة.



ملحوظة: بسبب تحكم الذاكرة الفائق، فهذا لا يشكل عقبة للـ Windows 95، 98، وخاصة الـ NT كما كان في الـ Windows 3.1.

يوضح شكل (٩-٨) عرض الـ TelName. لأفضل النتائج، إقصر العدد على ثلاثة أو أربعة أرقام قبل ضغط Names. توضح القائمة (٩-٥) source code للبرنامج. للحصول على شرح عن كيفية عمل الـ algorithm، وعن البدائل بشكل عام، أنظر مقالين "Algorithm Alley: Telephonic Mnemonics"، في صحيفة Dr. Dobb's Journal بتاريخ يونية ١٩٩٣.



شكل (٩-٨): يحول الـ TelName الصيغ الهاتفية الى أرقام هاتفية مساوية، ويمكنه أن يظهر كل البدائل الأبجدية الممكنة لأي رقم

القائمة (٩-٥): TelName\Main.pas

```
unit Main;

interface

uses
  SysUtils, Windows, Messages, Classes, Graphics,
Controls,
  Forms, Dialogs, StdCtrls, Buttons, Mask;

type
  TMainForm = class(TForm)
    Label1: TLabel;
      Label2: TLabel;
      ListBox1: TListBox;
      CloseBitBtn: TBitBtn;
      NamesBitBtn: TBitBtn;
      NumberBitBtn: TBitBtn;
      NameEdit: TEdit;
      NumbEdit: TEdit;
      procedure FormCreate(Sender: TObject);
      procedure NumberBitBtnClick(Sender: TObject);
      procedure NamesBitBtnClick(Sender: TObject);
      procedure NumbEditKeyPress(Sender: TObject; var Key:
    Char);
      procedure NameEditKeyPress(Sender: TObject; var Key: Char);
    private
    { Private declarations }
      function TelNameToNum(TelName: String): String;
      procedure ListNames(TelNum: String);
    public
    { Public declarations }
    end;

var
  MainForm: TMainForm;
```

```
implementation

{$R *.DFM}

var
{ Array of telephone dialog letters }
 TelDial: array[0 .. 9] of String;

{ Return telephone digit that corresponds to C }
function DigitToLetter(C: Char): Char;
var
 I, J: Integer;
begin
 C := Upcase(C);
   for I := 0 to 9 do
    for J := 1 to 3 do
       if (C = TelDial[I][J]) then
         begin
          Result := Chr(I + Ord('0'));
            Exit;
           end;
       Result := C;  { Default }
end;

{ Return ordinal value of digit character C }
function ValueOfChar(C: Char): Integer;
begin
 Result := Ord(C) - Ord('0');
end;

{ Return number for a telephone alphabetic name }
function TMainForm.TelNameToNum(TelName: String): String;
var
   I: Integer;
begin
```

```
  Result := '';
   for I := 1 to Length(TelName) do
    Result := Result + DigitToLetter(TelName[I]);
end;

procedure TMainForm.ListNames(TelNum: String);
var
 S: String;  { Temporary string }

   { Find N alphabetic permutations of digits in TelNum }
   procedure Permute(N: Integer);
   var
    I, Digit: Integer;
      begin
    Digit := ValueOfChar(TelNum[N]);
       for I := 1 to 3 do
       begin
         S[N] := TelDial[Digit][I];  { Insert letter }
           if (N = Length(TelNum)) then
            ListBox1.Items.Add(S)     { Add string to ListBox }
              else
            Permute(N + 1);  { Call Permute recursively }
            end;
      end; { Permute }

begin
   if Length(TelNum) > 0 then
   begin
    S := TelNum;    { Assign to temporary string }
       Permute(1);     { Start permutations }
      end;
end;

{ Initialize global variables }
procedure TMainForm.FormCreate(Sender: TObject);
begin
```

```
    TelDial[0] := '  '; TelDial[1] := '  ';
     TelDial[2] := 'ABC'; TelDial[3] := 'DEF';
     TelDial[4] := 'GHI'; TelDial[5] := 'JKL';
     TelDial[6] := 'MNO'; TelDial[7] := 'PRS';
     TelDial[8] := 'TUV'; TelDial[9] := 'WXY';
    end;

{ Do Number button click }
procedure TMainForm.NumberBitBtnClick(Sender: TObject);
begin
     NumbEdit.Text := TelNameToNum(NameEdit.Text);
     NumbEdit.SetFocus;
end;

{ Do Names button click }
procedure TMainForm.NamesBitBtnClick(Sender: TObject);
begin
     ListBox1.Clear;
     ListNames(NumbEdit.Text);
     ListBox1.SetFocus;
end;

{ Click Names button for Enter key in Number Edit object }
procedure TMainForm.NumbEditKeyPress(Sender: TObject;
  var Key: Char);
begin
  if Key = #13 then
     begin
      NamesBitBtn.Click;
          Key := #0;
        end;
end;

{ Click Number button for Enter key in Name Edit object }
procedure TMainForm.NameEditKeyPress(Sender: TObject;
  var Key: Char);
```

```
begin
    if Key = #13 then
    begin
     NumberBitBtn.Click;
        Key := #0;
      end;
end;

end.
```

يوضح الـTelName بعض واجــهــات التطبــيق المفــيــدة وتقنيــات string-handling. وهناك إثنتين من الـ labels توفران accelerator hot keys لـ Edit controls، باستخدام الـ methods الموضحه سابقاً. على سبيل المثال، تضع الـTelephone name label قبل الـn فى علامة الـ $، وخاصية الـ Control فى الـ Label محددة بـNameEdit. مع قيم هذه الخصائص، فإن ضغط Alt+N يختار الـ .labeled Edit object.

وتعرض أزرار الـ Names والـ Number الخاصة بالبرنامج كيفية تحويل الـ focus الى control آخر بعد اختيار الزر على سبيل المثال، عندما تضغط زر الـ Number يحول البرنامج الصيغة الهاتفية فى الـ Edit لاسم الـTelephone الى رقم. قد يريد المستخدمون تعديل هذا الرقم لذكر حروف اخرى ممكنة، لذا يقوم الـ OnClick الخاص بزر الـ Number بتحويل الـ focus بعيداً عن نفسه الى الـ NameEdit. تتعامل العبارتان التاليتان مع اختيار زر الـNumber:

```
NumbEdit.Text := TelNameToNum(NameEdit.Text);
NumbEdit.SetFocus;
```

تحول العبارة الأولى النص الخاص بـ NameEdit object، boat-Edit مثلاً- الى رقم الهاتف المساوى (2628). وتستدعى العبارة الثانية الـ SetFocus method الخاص بالـNameEdit لنقل الـ focus الى الـ Edit object التالى.

ان ضغط زر الـ Names يؤدى مهمة مشابهة باستخدام هذه العبارات الثلاث فى OnClick:

```
ListBox1.Clear;
```

```
ListNames(NumbEdit.Text);
ListBox1.SetFocus;
```

أولاً، يمسح الـ procedure المدخلات الحالية للـ ListBox. ثم يستدعى بعد ذلك الـ ListNames الذى يبدل رقم هاتف بكل الصيغ الابجدية الممكنة. وتحول العبارة الاخيرة الـ input focus الى الـ ListBox حتى يمكن للمستخدمين ضغط مفاتيح الاسهم لرؤية النتائج الواردة.

ان توفير عمليات لوحة مفاتيح فعالة للبرامج الجرافيكية يعتبر سحراً اسود لا يمارس بشكل جيد عادة. بالإضافة الى السماح بتحكم لوحة المفاتيح على الأوامر، تستخدم واجهة تطبيق المستخدم بشكل جيد methods مثل الـ SetFocus لتقليل التعامل مع المفاتيح. بالرغم من انه من المأمون ان نقول ان كل نظم الـ Windows لديها وسيلة ادخال بالفأرة، فالتطبيق التام يجب لن يكون من الممكن تنفيذ عملياته من لوحة المفاتيح.

الـ TelName يتخذ procedures آخرين لتحسين التعامل مع لوحة المفاتيح. عندما تضغط Enter فى الـinput focus عادة بالتحويل الى الـ control التالى فى ترتيب الـ Tab. هذا يوفر فرصة ممتازة لأداء ايه عمليات لازمة على محتويات الـ Edit control، مما يزيل ضغطه زر واحدة. قم بتشغيل الـ TelName وإدخل boat فى الـ Edit control لاسم الـ Telephone. عندما تضغط Enter، يحول البرنامج الـ focus الى منطقة الـ Edit برقم الـ Telephone، ويحول الـ boat الى الرقم المساوى لها. اضغط Edit مرة اخرى للتحول الى الـ ListBox وإدخل كل البدائل الممكنة لهذا الرقم. عندما تجرب هذا، سيبدو والتأثير واضحاً، ولكن التوصل الى جهاز لوحة مفاتيح بصورة سهلة يتطلب عادة مزيداً من التفكير والاختيار.

لإعادة برمجة مفتاح الـ Edit لتنفيذ هذه المهام المتفاعلة، يستخدم الـ TelName إثنين من الـ OnKeyPress event handlers للـ NameEdit objects والـ NumbEdit. فى الـ NumbEditKeyPress، مثلاً، يبحث البرنامج عن carriage return (يمثل التعبير #13 رمز ASCII بقيمة ١٣). لمعالجة المعلومة فى نافذة الـ Edit، يحاكى البرنامج ضغطه زر بعبارة:

```
NamesBitBtn.Click;
```

بعـد ذلك، يلقى البرنامج بالـ carriage return لان الزر يحـول بالفـعل الـ focus الى نافـذة الـ Edit التـاليـة، ونحـن لا نريد ان يتلقى الـ control الحـالى هذا الرمز. للتخلص من ضغطة مفتاح الـ Edit، حـدد الـ Key بـ ASCII لإلغاء هذا التعيين:

```
Key := #0;
```

الـ procedure ListNames يستدعى الـ Permute procedure المتداخل لايجاد كل المجموعات الابجدية لرقم تم تمريره كـ TelNum parameter.

:Masked Edit boxes

يوفر الـ MaskEdit component هيئة الـ Edit والذى يحدد للـمسـتـخدمين نوعـيـة الرمـوز المطلوبة ادخـالهـا فى مكان مـحـدد. وهناك secondary string، يدعى الـ EditMask، يحدد قواعد الادخـال. على سبيل المثال، يمكن ان يـحدد الـ mask الـ format الخـاصـة برقـم هاتف والذى يمدد المسـتـخـدمين ان يدخلوا رقم هاتف مكون من سبعة ارقام. اتبع هذه الخطوات لتجربة الـ MaskEdit:

١- أضـف الـ MaskEdit مـن الـ Additional VCL palette عـلى الـ form. وكذلك أضف الـ Edit object و Button.

٢- يقـدم الـ MaskEdit خـاصـيـة، وهى الـ EditMask، تحـدد تنسـيق الادخال الخاص بالـ control. يمكنك تعيين قناعاً فى وقت التصـميم بتعديل هذه الخاصية. ولفعل نفس الشئ فى وقت التشغيل، إضغط الـ Button1 مرتين وادخل هذه العبارة فى OnClick الخاص بها:

```
MaskEdit1.EditMask := Edit1.Text;
MaskEdit1.SetFocus;
```

٣- قم بتـشـغـيل البـرنامج وإدخل 999\-9999;1;_ فى الـ Edit1 object. إدخل كل علامات الترقيم كما هو موضح (لاحظ underscore الأخيرة). اضغط الزر لنسخ هذا النص الى خـاصـيـة الـ EditMask لـ MaskEdit. يـحـدد الـ mask رقم هاتف من سبعة ارقام (سوف اشرح المزيد عن تنسيق الـ mask فيما بعد).

٤ - إدخل نصاً فى الـ MaskEdit لتجربة الـ mask .

يمكنك ايضاً إختيار الـ mask بضغط الزر البيضاوى لخاصية الـ EditMask لـ MaskEdit . هذا يفتح الـ Input Mask Editor dialog الخاص بـ Delphi، الموضح فى شكل (٩-٩) اختر mask، وإدخل strings اختبارية فى الـ Test Input edit control . قم بتحميل ملفات الـ dem . (Delphi Edit Mask) بضغط زر الـ Masks . يقدم Delphi ملفات عديدة ذات ارقام هاتف، تواريخ، اوقات، و formatting masks اخرى بعدة لغات. توضح القائمة (٩-٦) نصاً من الـ Denmark.dem، الموجود فى الدليل الفرعى Bin الخاص بـ Delphi .

شكل (٩-٩): الـ Input Mask Editor dialog الخاص بـ Delphi يقدم mask لخاصية الـ EditMask التابعة لـ MaskEdit

القائمة (٩-٦): يوضح الـ Denmark.dem الـ format للف ask text

Phone | 48140001 | 00 09 99 99;1;_
Phone with Country | 48140001 | \+45 00 00 00 00;1;_
Social Security | 1234567890 | 000000\-0000;1;_
Zip Code | 3450 | 0000;1;_
Zip Code with Country | DK3450 | !>LL\-0000;1;_
Date | 260195 | 90\.90\.\1\900;1;_
Date (Windows) | 260195 | 90/90/00;1;_
Long Time | 210515 | !90:00:00;1;_
Short Time | 1345 | !90:00;1;_

للحصول على جدول بالرموز المكتوبة التى يمكنك استخدامها لإنشاء masks، أبحث عن الـ MaskEdit فى online help الخاصة بـ Delphi . يعد الـ EditMask string من الرموز التى تحدد الرموز الحرفية، مجموعات رموز

الادخـال، وشـفـرات التـحكم. يتكون الـ mask من ثلاثة اجـزاء منفـصلة بفـصلة منقوطة:

- رمزاً واحداً أو أكثر يحدد مجموعاً رموز الادخال المسموحة.
- الرقم 1 لحفظ الرموز الحـرفـيـة فى الادخـال- hyphens فى رقم هاتف مثلاً- أو الرقم (0) حتى لا يحفظ الرموز الحرفية.
- رمـزاً لعرض للاماكن الخـاليـة فى نافـذة الـ MaskEdit- هذا يكون عـادة underscore أو blank.

توضح EditMask string كيفية إنشاء كل جزء من الـ mask الكامل. ها هو الـ mask:

!\(999\)000\-0000;1;_

فى وقت التـشـغـيل، يجـعل هـذا الـ mask الـ MaskEdit يعـرض قـالب الادخال التالية:

(___)___-____

فى الـ mask string ، تخبر نقطة تعـجب رأسية الـ control ان يطرح جانباً المسـافـات المتـتـابعـة من الادخـال الناتج. اذا لم تبـدأ الـ mask بنقطة تعـجب، يتم المحافظة على المسافات المتتابعة. اكتب back slash الى الخلف لتحدد رمزاً حرفياً- هناك ثلاثة فى هذا المثال: أقـواس الفتح والإغلاق حـول code مـساحـة الارقـام الثـلاث الخـاصـة بالرقم، و hyphen التى تفـصـل الـ exchange والرقم. يظهـر الـ control هذه الرموز فى نافذة للـ edit، وتمر عليها text cursor مروراً خاطفاً اثناء الكتابة تلقائياً.

والـ 9 فى الـ mask تشير الى ان المستخدمين بامكانهم إدخال رقماً أولا شئ فى هذا المكان. استخدام الـ 0 لتطلب رقماً. استخدم حرف الـ A الكبير للسماح بالمدخل الهجائى الرقمى؛ استخدم حرف الـ C الكبير لتطلب رمزاً. انظر online help الخاصة بـ Delphi لتعرف رموز الـ mask اخرى.

والجزء الثانى من edit mask (1; فى المثال السابق) يشير اذا ما تحفظ الرموز الحرفية فى الادخال الناتج. إدخل 0; الى الرموز الحرفية المتروكة. مع هذا الخيار،

يتلقى البرنامج رقم هاتف مثل الـ 1212-555(800) كـ string للارقام 8005551212.

والجزء الثالث والاخير من edit mask تحدد الرمز الذى يجب عرضه لمواضع الادخال فى قالب الـ mask عادة، يجب ان تحدد underscore أو مسافة خالية، ولكن يمكنك استخدام رمزاً آخر اذا اردت. على سبيل المثال، قم بتعيين الـ underscore المتتابعة الخاصة بالـ mask السابق الى نقاط.

تعتبر النقاط ورموز علامات الترقيم الاخرى Fonts متناسبة تنتهى بقوالب الـ MaskEdit التى تشغل مسافة اقل من الادخال النهائى. للحصول على قالب ذا شكل افضل عند استخدام النقطة كرمز موضع الادخال، اختر Font احادى المسافة مثل الـ Courier New للـ MaskEdit control.

يمكنك إنشاء ملفات الـ dem. الخاصة بك لقوالب الـ mask للتحميل فى الـ Input Mask Editor dialog الخاص بـ Delphi. وهذا هو مجرد ملف نص عادى ذو امتداد اسم الملف dem. يتكون كل سطر فى الملف من ثلاثة اجزاء: اسم الـ mask، inputs، و edit-mask string. على سبيل المثال، ها هو مدخل الـ Zip Code من الـ Denmark.dem:

Zip Code | 3450 | 0000;1;_

ملحوظة: يقدم Delphi ملفات dem. لمختلف البلدان واللغات المحلية. تحتوى هذه الملفات على mask للادخال تتبع الـ format الخاصة بالدول. للعثور على الملفات، إبحث فى دليل التركيب bin\ الخاص بـ Delphi.

الـ Password entry :

يمكنك استخدام الـ Edit أو MaskEdit لتوفير password-entry box (وهذا ايضاً ينطبق على الـ DBEdit لقاعدة البيانات). قم بتصميم أى رمز غير الـ 0# فى خاصية الـ PasswordChar. على سبيل المثال، إدخل * أو # فى هذه الخاصية فى الـ Edit وقم بتشغيل البرنامج. يمكن للمستخدمين إدخال نص فى الـ control، ولكن رمز كلمة المرور فقط هو الذى يظهر على الشاشة.

أعط المستخدمين دائماً الفرصة لتأكيد كلمة المرور بكتابتها مرتين . ان واحدة من اسهل الطرق لفعل هذا هى بإضافة اثنين من لـ Edit أو MaskEdit على الـ form. وهذه الـ format قد يكون dialoge يعرض أمر، أو يظهره البرنامج عند البدء قبل السماح بالتوصل الى المعلومات .

قم بإنشاء الـ OnCloseQuery الخاص بالـ form . تعتبر القائمة (٩-٧) مثالاً على برنامج يقارن بين الـ Edit controls . يسمح البرنامج للـform بالإغلاق فقط اذا كانت كلمتى المرور متناسبتين . عند استخدام هذا البرنامج استخدم اثنين من الـ Edit والـ OnCloseQuery لمنع الـ form من الإغلاق الا اذا كانت كلمتى السر فى الـ controls متشابهتان .

القائمة (٩-٧)؛مقارنة اثنين من الـ Edit controls

```
. TS: The following line of code exceeds 64 characters in
length. Please break the line. Thanks. - DSprocedure
TForm1.FormCloseQuery(Sender: TObject;
  var CanClose: Boolean);
begin
    if Edit1.Text <> Edit2.Text then
    begin
     ShowMessage('Password incorrect!');
       CanClose := False;
       Edit1.SetFocus;
      end;
end;
```

أو، يمكنك إدخال Password Dialog form فى تطبيق. إتبع هذه الخطوات لتجربة هذا القالب:

١- ابدأ مشروعاً جديداً، اختر File|New Form أو اضغط زر الـ New . اختر Password Dialog من الـ Browse Gallery dialog .

٢- أضف Button على الـ form الرئيسية . اضغط الزر مرتين وادخل هذه العبارات فى OnClick الخاص بالـ Button :

```
PasswordDlg.ShowModal;
ShowMessage(PasswordDlg.Password.Text);
```

٣- أضف امر الـ uses الخاص بالـ Unit2 الى ذلك الخاص بالـ Unit1 قرب قمة source code module الخاصة بـ Unit1 .

٤- قم بتشغيل البرنامج واضغط الزر . إدخل كلمة المرور فى dialog box . عندما تضغط OK يعرض البرنامج مدخلاتك فى ShowMessage dialog . لاحظ ان البرنامج المضيف يتوصل الى نص كلمة المرور الخاص بالـ dialog على انها التعبير PasswordDlg.Password.Text. (تعتبر كلمة المرور Edit object تمتلكه الـ PasswordDlg) .

افكار للمستخدم الخبير

● للحفاظ على مسافات التخزين ، ضع قبل function string parameters و الـ procedure كلمة const . هذا يخبر Delphi أن الـ procedure يستخدم الـ string ولكنه لا يغيره . وبالتالى ، يمكن ان يمرر Delphi الـ string ، الذى يهدر مسافات التخزين . (ان الـ string ليست ذاكرة تخزين مخصصة فى النسخة الحالية من Delphi ، ولكن الـ const لا يزال يحفظ string references خفى فى الـ try-finally ، والتى يمكن ان تنشئ اطارات تخزين كبيرة) . على سبيل المثال ، عرف procedure كما يلى لإنشاء string parameter ثابت :

```
procedure MySubroutine(const S: String);
begin
end;
```

● حدد الـ ShowAccelChar الخاصة بالـ Label بـ False لإبطال قدرة الـ Label على تزويد accelerator key للـ controls الاخرى . قد تريد ان تفعل هذا ، مثلاً ، لعرض علامة الـ & فى الـ Caption الخاص بالـ Label . ولكن يمكنك عرض الـ & ببساطة بكتابة إثنين منها . ويتم عرض الـ & ببساطة بكتابة إثنين منها . ويتم عرض الـ &&Amp Label Caption على انه &Amp .

● لقصر الادخال من خلال الـ Edit control على حروف الكبيرة أو الحروف الصغيرة ، بدلاً من كتابة الـ OnKeyDown event الخاص بالـ control ،

اخـتـر قـيـمـة لخـاصـيـة الـ CharCase . يمكنك تحـديد الـ CharCase بـ ecLowerCase ، ecNormal ، أو ecUpperCase .

● ان الـ Label ليس لها window handles ولذا فهى تستهلك ذاكرة اقل من الـ Edit . للحفاظ على الذاكرة ، استخدم دائماً الـ الـ Label بدلاً من الـ Edit لنص القراءة فقط .

● يعـرض الـ MessageDlg والـ MessageDlgPos عـادة glyph bitmaps على ازرارهم . اذا لم تكن تريد glyphs على الازرار ، إدخل العـبـارة التالية فى OnCreate الخاص بالـ form الرئيسية :

```
MsgDlgGlyphs := False;
```

● حدد خاصية الـ Transparent الـ Label بـ True لعرض text labels فوق الصور الجرافيكية مثل الـbitmaps .

● ان Label لا يعـد windowed control ، ولكونه هكذا ، فـلا يمكن ان يظهر فوق windowed control مثل الـ Button . اذا لم يكن بامكانك ان تجبر الـ Label على الظهور فوق component آخر باوامر الـ Edit|Bring to Front أو Edit|Send to Back ، فإنك تحاول المستحيل . جرب وسيلة اخرى للحصول على التأثير الذى تريده ، أو استخدم الـ TStaticText component. .

● ان خاصية الـ Modified التابعة لـ Edit component's ، وهى قـيـمـة Boolean محددة بـ True أو False ، تشير اذا ما حدث اية تغييرات لمحتويات الـ control منذ إنشاء الـ Edit أو منذ تحديد الـ Modified بـ False . بعـد قـبـول الادخال من الـ Edit ، حدد الـ Modified بـ False . يمكنك حينئذ فحص هذه الخاصية لتحدد ما اذا كانت محتويات الـ Edit قد تغيرت .

● يمكنك التوصل الى نص الـ MaskEdit باستـخـدام خـاصـيـتين ، الـ EditText أو الـ Text . اذا كان object's mask يحدد ان الرموز الحرفية يجب ان يتم حفظها ، فإن الـ EditText و الـ Text لهـمـا المحتويات المتطابقة ، ويمكنك استخدام أى منهما للحصول على control's text . اذا لم يكن الـ mask يحدد حفظ الرموز الحرفية ، فإن الـ EditText تمثل النص الذى يراه المستخدمون فى نافذة الـ MaskEdit ، و الـ Text تمثل نفس هذا النص ناقص رموزه الحرفية .

● لقصر الادخال من خلال الـ MaskEdit على الحروف الكبيرة والحروف الصغيرة، حدد خاصية الـ CharCase بـ ecLowerCase أو ecUpperCase. للسماح بالادخال بالاحرف الصغيرة والحروف الكبيرة، حدد الـ CharCase بقيمتها الافتراضية ، ecNormal. قد يكون هذا ابسط من تحديد دخول الحروف الصغيرة وحروف الكبيرة باستخدام رموز الـ الـmask.

● عند تعاقب الـ strings، تذكر ان كل استخدام لعامل الزائد يستهلك ذاكرة للتخزين المؤقت. فى كثير من الحالات، يمكنك استخدام الـ Format function لتقليل استخدام الذاكرة. على سبيل المثال، لتعاقب ثلاث strings A، B، C، وتعيين الناتج الى S، استخدم عبارة مثل :

```
S := Format('%s%s%s', [A, B, C]);
```

وهذا هو ما يعادله منطيقياً للعبارة التالية :

```
S := A + B + C;
```

المشروعات التى يمكنك تجربتها

(٩-١) : قم بتصميم module تعرض رسائل خطأ باستخدام الـ MessageDlg أو الـ MessageDlgPos .

(٩-٢) : اكتب برنامجاً اختيارياً يسمح للمستخدمين ببرمجة accelerator hot keys لواحد أو أكثر من الـ Edit objects .

(٩-٣) : قم بإنشاء MaskEdit objects لإدخال عناوين البريد الالكترونى لخدمات عديدة مثل الـ Internet والـ CompuServe . إحفظ قوالبك فى ملف نص dem. .

(٩-٤) : اكتب password entry dialog أو قم بالتحسين فى قالب الـ Password Dialog الخاص بـ Delphi . يجب ان يقوم قالبك بتوفير السبل للمستخدمين ليؤكدوا كلمة المرور وذلك بإدخالها مرتين .

ملخص:

ان الـ Label، والـ Edit ، والـ MaskEdit components الخـــاصـــة بـ Delphi يمكن ان تتعامل مع غالبية متطلبات single-line string بالتطبيق .

• يقـــدم Delphi نوعين اســـاسين من للـ strings-AnsiString والـ ShortString . ان نوع بيانات الـ strings يرجع اصلاً الى الـ AnsiString الا اذا كـان يسـتـخـدم خـيـارا {-$H} ، فى هذه الحـالة تكون الـ Strings من نـوع الـ ShortString . تعتبر الـPascal strings ذات الـ ٢٥٥ رمزاً بطول البايت مساوية للـ ShortString . ان انواع بيانات الـString (و الـ AnsiString) تنشئ strings ديناميكية غير محددة الطول . يتم التحكم الآلى فى الذاكرة عند استخدام هذه الانواع من الـ strings . ان {-$H} ، يعـتـبـر مـفـيـداً فـقط لإجـراء عـمليـة الـ compiling للـ Pascal code والـ Delphi القديمة .

• يقدم Delphi ايضاً نوع الـ WideString ، الذى يخزن قيم رموز ذات ١٦ بت Unicode . ان strings العريضة يتم استخدامها فى تطبيقات اللغات مثل اليابانية ، التى لا يمكن ان تمثل رموزها كاملة باستخدام البايت ذات الـ ٨ بت . وهى ايضاً تسمح بالعرض الغير غامض للنص بلغات متعددة .

• حاول استخدام الـ Pascal Strings لجميع احتياجاتك للتعامل مع الـ strings . الـ functions والـ procedures للـ Pasc string فعالة جداً ، ولان الـ string من النوع المركب داخلياً ، يمكنك استخدام string فى تعبيرات . على سبيل المثال ، لتعاقب إثنين أو أكثر من الـ strings ، اضفهم الى بعضهم البعض بعوامل علامة الزائد ، كما فى التعبير .S1 + S2 + S3 يمكنك ايضاً مقارنة الـ Pascal Strings باستخدام عوامل الـ<، > ، => ، =<، و<> = . يجب ان تستدعى functions ومثل الـ StrComp لمقارنة الـ null-terminated strings .

• لعـــرض رســـائلك اســـتـــدع الـ ShowMessage أو الـ ShowMessagePos . لعرض رسائل وحث المستخدمين على ضغط زر- مثلاً ، لطلب الأذن بحذف ملف- إستدع الـ MessageDlg أو الـ MessageDlgPos . يمكنك ايضاً استدعاء الـ MessageBox function المعيارية بالـWindows ، ولكن هذا يتطلب منك استخدام null-terminated strings .

● لحث المستخدمون على ادخال بيانات نص احادى السطر ، استدع الـ InputBox أو الـ InputQuery . هذه الـ functions يمكن ان تتعامل مع اغلب إحتياجات الادخال بسطر واحد ، وتوفر عليك الوقت والجهد فى تطوير input dialogs الخاصة بك مع الـ Edit component .

● كما تعرف غالباً يمكنك استخدام الـ Label object وعناصر النص الثابت الاخرى فى الـ forms . ولكن الـ Label object يمكن ايضاً ان يوفر accelerator hot key لـ control آخر مثل الـ Edit أو الـ MaskEdit الذين لا يقدمون اختيار مفتاح Alt .

● لاحتياجات ادخال السطر الواحد العامة ، استخدم الـ Edit object . وهذا الـ components له events وخصائص عديدة يمكنك استخدامها لتوفير مناطق ادخال . للـ Edit عبارة عن text editor محرر نص احادى السطر ، ويدعم تلقائياً عمليات الفأرة ولوحة المفاتيح ، بما فى ذلك القص ، النسخ ، واللصق .

● حاول ان تجعل تطبيقاتك قابلة للعمل بالفأرة ولوحة المفاتيح . على سبيل المثال ، قم بتشغيل تطبيق الـ TelName ، والذى يظهر تقنيات جهاز لوحة المفاتيح مثل تحديد الـfocus عند تلقى ضغطه زر .

● استخدم الـ MaskEdit لتوفير templeete-based Edit controls . ادخل أو اختر mask لخاصية الـ EditMask لـ MaskEdit object لتطلب من المستخدمين ، مثلاً ادخال رقم هاتف أو تاريخ باستخدام form معينة .

● لإدخال كلمة المرور ، عين رمزاً لخاصية الـ PasswordChar لـ Edit أو الـ MaskEdit object (وهذا ايضاً يعمل مع الـ DB-Edit components) . تعرض الـ objects هذا الرمز فقط لكل المدخلات .

لقد بدأت الآن فقط تتعرف على إمكانيات الادخال والإخراج الخاصة بـ Delphi . فى الباب التالى ، سوف تفحص components للتحكم فى الـ text objects. متعدد السطور .

*****



الباب العاشر
العمل مع نص متعدد السطور

محتويات هذا الباب:

- Components
- الـ Memos
- Text and the clipboard
- Scrolling down the river
- StringGrids

سوف يبدأ المبرمجون، آجلاً أو عاجلاً، فى كتابة text editors الشخصى الخاص بهم بكل الميزات التى يحلمون باستخدامها. ان تصميم text editors بسيط بأقل قدر من اوامر نص، قص ولصق السطور، وقراءة وكتابة الملفات كل هذا يعد أمراً بالغ المشقة. وانا لم اكمل ابداً text editor الخاص بى، واتمنى ألا اشرع فى إنهاءه.

ولحسن الحظ، مع الـ multiple-Line text components الخاص بـ Delphi، إننى لا احتاج غالباً الى العمل على هذا الـ code مرة اخرى. ان Memo واحد يوفر كل إمكانات الـ Windows Notepad editor. فقط ادخل الـ Memo فى form وحرك عصاك السحرية لتحصل على text editor على الفور.

فى هذا الباب، إنك تستخدم الـ Memo والـ StringGrid لإقامة text objects متعدد السطور فى واجهة تطبيق المستخدم لتطبيقك. وسوف اشرح ايضاً موضوعات متعلقة بهذا ايضاً مثل إنتقال المعلومات خلال الـ clipboard وكيفية استخدام الـ ScrollBar والـ ScrollBox ، واللذان سوف تحتاجها دائماً عند العمل بنص متعدد السطور.

: Components

فيما يلى Delphi components للعمل بنص متعدد السطور :

• **Memo**: من الناحية العملية برنامج معالج كلمات، يضيف الـ Memo إمكانات تشبه الـ Notepad الى التطبيق. ان متصفح ملف نص الـ Readme الخاص بهذا الباب- والذى يمكنك استدعاه من تطبيق آخر installation utility- يوضح كيفية استخدام الـ Memo objects . الـ Palette : Standard .

• **ScrollBar**: هذا الـ component يضيف scroll bars للـ form، ويمكنه ان يعمل مستقلاً كـ range-selection control. على سبيل المثال، فى هذا الباب، إنك تستخدم الـ ScrollBar لإنشاء color selection dialog . الـ Palette : Standard .

• **ScrollBox** : يمكنك ان تسمى هذا الـ component بالـ object الحاوى ذى العجلات. استخدم الـ ScrollBox لإنشاء scrollable objects، panels، text displays ، options dialogs، وعناصر بيئة معقدة اخرى تحتاج الى إمكانات scrolling . الـ Palette : Additional .

• **StringGrid**: هذا الـ component ذو القدرة العالية، والذى تم تقديمه فى الباب السابق، ينظم string lists فى الـ form فى صورة اعمدة وصفوف يمكن التوصل إليها كـ array ثنائى الابعاد. مع هذا الـ component ، لديك اغلب العناصر البيئة اللازمة لإنشاء جداول ودفاتر حسابات. ان تطبيق الـ CharGrid الخاص بهذا الباب، والذى يشبه الـ Windows Character Map utility، يوضح كيفية استخدام الـ StringGrid objects لإنشاء دفاتر حسابات قابلة للاختيار. الـ Palette : Additional .

بفضل الـ Memos:

يغلف الـ Memos الـ Edit control متعدد السطور المعيارى بالـ Windows. ولكن الـ Memos ليس control قديماً فى زى جديد- إنه text editor معقد متعدد السطور يمثل ميزات object-oriented programming . على سبيل المثال، ان الـ Edit control المعيارى بالـ Windows يجعل التحكم فى

حواجز تخزين النص امر صعب وميال للخطأ . ان الـ Memos ، الذى يغلف الـ Edit control فى الـ TMemo) Delphi class)، و يهـــزم هذه الحـــواجـــز الـmethods المصممة بذكاء للتوصل الى نص متعدد الاسطر فى forms حواجز تخزين أو string-list .

إنشاء Memos للقراءة فقط:

ان الـ Read-only Memo مفيدة فى عرض المعلومات . على سبيل المثال ، انظر تطبيق الـFancy فى الباب السادس ، الذى يستخدم إثنين من الـ Memos لعرض قيم خاصية لانماط الـ Panel 3D المتنوعة . ان استخدام Memos لقوائم القراءة فقط اسهل بكثير من إنشاء Labels متعددة .

لإنشاء Memos للقراءة فقط ، حدد ثلاث خصائص للقيم الواردة فى جدول (١٠-١).

جدول (١٠-١): خصائص الـ Memos للقراءة فقط

المواصفات	القيمة	الوصـــف
Enabled	False	تمنع المستخدمون من اختيار نص فى نافذة الـMemo
TabSrep	False	وتمنع المستخدمون من ضغط الـMemo
ReadOnly	True	تمنع المستخدمون من عمل إية تغييرات أو إدخال نصاً جديداً فى نص الـMemo

بعد تحديد هذه القيم ، عين نصاً لخاصية الـ Lines لـ Memo . اضغط الزر البيضاوى لفتح محور قائمة الـ String الخاصة بـ Delphi، واكتب أو انسخ والصق نصك .

فى النسخ ذات الـ 16-bit من Delphi ، يمكن أن يحمل الـ Memo الى 32K من النص . كل سطر فى الـ Memo لا يمكن أن يكون اطول من 1,024 رمـــزاً، شريطة ألا يكون عرض السطر المعروض اطول من 30,000 pixels ومواصفات الـ Windows هذه تم إزالتها من النسخ ذات الـ 32-bit من Delphi ، ويمكن الآن أن تحمل الـ Memo كم غير محدد من النص ، بالرغم من أن الـ Windows 95 مازال يحدد الكم الإجمالى بـ 64K .

إخراج وإدخال ملف النص:

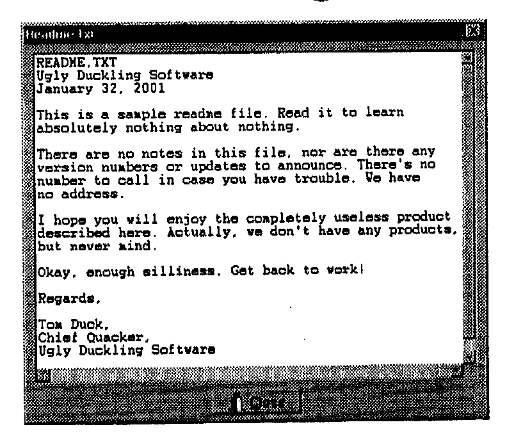
تعتبر خاصية الـ Lines للـ TStrings object Memo، ولذلك، يمكنك استخدام تقنيات string-list من الباب الثامن لقراءة، وكتابة، وتعديل نص الـ Memo. على سبيل المثال، هذا يعرض السطر الأول فى الـ Memo:

```
ShowMessage(Memo1.Lines[0]);
```

استدع الـ LoadFromFile method الخاص بالـ Lines لقراءة ملف النص فى الـ Memo. استدعى الـ SaveToFile لكتابة محتويات الـ Memo على قرص. الـ procedures، يمكنهما إقامة الـ editor الشبيه بالـ Notepad الخاص بك.

On the CD-ROM

على القرص المدمج: كمثال على التعامل مع ملف نص، يعرض تطبيق الـ Readme نسخة قراءة فقط للملف نص فى الـ Memo يمكن لبرنامج آخر- installation utility، مثلاً- أن يستدعيه لعرض ملاحظات حول التطبيق بدلاً من استخدام الـ method القديم وهو استدعاء الـ Notepad، والذى يمكن المستخدمين من تغيير ملف القراءة فقط الخاص ببرنامجك. بعد قائمة الـ Readme، سوف اشرح كيفية تشغيل البرنامج من داخل تطبيق آخر لـ Delphi- مثلاً، يمكنك استخدام التقنية فى برنامج installer. يوضح شكل (١٠-١) عرض الـ Readme. توضح القائمة (١٠-١) الـ source code، الموجودة على القرص المدمج فى دليل الـ Source\Readme.



شكل (١٠-١): يوضح تطبيق الـ Readme كيفية إنشاء متصفح ملف نص قراءة فقط

القائمة (١٠-١): Readme\Main.pas

```
unit Main;

interface

uses
 Windows, SysUtils, Messages, Classes, Graphics,
   Controls, Forms, Dialogs, StdCtrls, Buttons;

type
   TMainForm = class(TForm)
    Memo1: TMemo;
       BitBtn1: TBitBtn;
       procedure FormActivate(Sender: TObject);
     private
    { Private declarations }
     public
    { Public declarations }
     end;

var
   MainForm: TMainForm;

implementation

{$R *.DFM}

procedure TMainForm.FormActivate(Sender: TObject);
var
 FileName: string;
begin
 if ParamCount >= 1
     then FileName := ParamStr(1)
```

```
        else FileName := 'Readme.Txt';
          Memo1.Lines.LoadFromFile(FileName);
        Caption := FileName;
  end;

  end.
```

يوضح OnActivate للـ form الرئيسية للـ Readme كيفية الحصول على parameter تم تمريره من عملية أخرى الى تطبيق. (أدخل parameters إختبارية بأمر الـ Run|Parameters). إن المتغير العام ParamCount يساوى عدد parameters، كلاً منها متاح من خلال الـ ParamStr function. قم بتمرير الـ indexed parameter الى الـ ParamStr. والـ indexed parameter بـ (0) يساوى اسم مسار التطبيق- قم بعرضه فى OnClick الخاص بالـ Button بعبارة:

```
ShowMessage(ParamStr(0)); {Show application pathname}
```

فى الـ Readme، اذا كان الـ ParamCount أكبر من أو يساوى واحد، يعين البرنامج parameter قد تم تمريره الى الـ FileName string، وإلا، فإنه يحدد الـ FileName للملف الإفتراضى، بـ Readme.Txt وتقوم عبارة واحدة بتحميل نص هذا الملف فى الـ Memo:

```
Memo1.Lines.LoadFromFile(FileName);
```

بعد ذلك، يحدد البرنامج الـ Caption الخاص بالـ form بـ FileName string. عند استخدام الـ Memo لعرض ملفات نص، استخدم Label، Caption، أو text object آخر لعرض اسم المسار حتى يعرف المستخدمون أى الملفات يتصفحون.

بالرغم من أن الـ Readme لا يكتب على قرص، يمكنك أن تفعل هذا بسهولة باستدعاء الـ SaveToFile procedure لخاصية الـ Lines لـ Memo. يقوم الـ procedure بإنشاء ملف جديد، أو ملف يكتب على ملف اذا كان يوجد واحداً يحمل هذا الاسم. للحماية من الكتابة على ملف موجود، استدع الـFileExists قبل استدعاء الـ SaveToFile، كما فى هذا الـ code (بافتراض ان الـ FileName يعتبر متغير String):

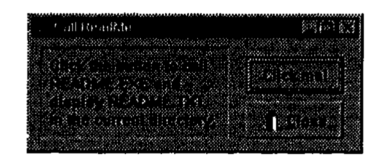
```
  FileName := 'C:\Anyfile.Txt';
  if FileExists(FileName) then
    ShowMessage('File exists')
  else
    Memo1.Lines.SaveToFile(FileName);
```

لتمرير الـ parameters الى برامج، استخدم أمر الـ Run لزر الـ Start بالـ Windows 95 أو 98 إدخل C:\Path\Readme Filename.txt لتشغيل الـ Readme وتمرير الـ Filename.txt parameter، إليه عند تشغيل برامج داخل Delphi، اختر أمر الـ Run|Parameters، وإدخل واحداً أو أكثر من الـ parameters لتمررها الى البرنامج فى المرة التالية التى تشغله فى نمط إزالة الأخطاء.

على القرص المدمج: بالرغم من انه بإمكانك تشغيل الـ Readme كتطبيق مستقل، لقد صممته على ان يتم استدعاءه من برنامج آخر، مثل installation utility. لعرض هذه التقنية، التى يمكنك استخدامها لتشغيل أى برنامج تنفيذى - فهى تعمل ايضاً للملفات الـ DOS PIF- اختبر تطبيق للـ Readme الموجود على القرص المدمج المرفق بهذا الكتاب فى دليل الـ Read me قم بتحميل البرنامج فى Delphi واضغط F9 للتشغيل. إختر زر الـ Click Me لشغيل تطبيق الـ Readme. يوضح شكل (١٠-٢) عرض الـ Readme. لتشغيل برنامج الـ Readme، اختر زر الـ Click Me [انظر شكل (١٠-١)]. توضح القائمة (١٠-٢) الـ source code، الموجودة فى دليل الـ Source\Readme.

ملحوظة: اذا لم يعرض الـ Readme ملف الـ Readme.txt المتوقع، إتبع هذه الخطوات. قم بتحميل ملف الـ Runme.dpr فى Delphi واضغط Ctrl+F9 لإجراء عملية الـ compile. ثم قم بتحميل Runme.dpr واضغط F9 . كلا المشروعين موجودان على القرص المدمج فى دليل الـ Source\Readme. يجب ان يكون الآن قادراً على ضغط الـ Click Me لشغيل برنامج عرض الـ Readme. اذا واجهت مشكلة، فالسبب غالباً هو ان البرنامج يتوقع ان يجد الـ Readme.txt فى الدليل الحالى، والذى قد تكون قمت

بتغيره اثناء استخدام Delphi . ان إعادة تشغيل كل برنامج عرض يضمن ان الـ Readme.exe يمكن ان يجد ملف الـ Readme.txt .



شكل (١٠-٢): يوضح تطبيق الـ Runme كيف يمكن لتطبيق ان يقوم بتشغيل برنامج آخر

القائمة (١٠-٢): Readme\Test.pas

```
unit Test;

interface

uses
  Windows, SysUtils, Messages, Classes, Graphics,
     Controls, Forms, Dialogs, ExtCtrls, StdCtrls, Buttons;

type
  TTestForm = class(TForm)
      ClickMeButton: TButton;
        Label1: TLabel;
        Bevel1: TBevel;
        BitBtn1: TBitBtn;
        procedure ClickMeButtonClick(Sender: TObject);
       private
      { Private declarations }
       public
      { Public declarations }
       end;

var
```

```
   TestForm: TTestForm;

implementation

uses FMXUtils;

{$R *.DFM}

(* 16-bit version; still works but WinExec is obsolete
procedure TTestForm.ClickMeButtonClick(Sender: TObject);
var
    K: Integer;  { Result of calling WinExec }
begin
  K := WinExec('Readme.Exe Readme.Txt',
   SW_SHOWNORMAL);
    if K < 32 then
      ShowMessage('Error running README.EXE');
end;
*)

{ 32-bit version; calls ExecuteFile in FMXUtils unit, which
  is located in Delphi's Demos\Doc\Filmanex folder }
procedure TTestForm.ClickMeButtonClick(Sender: TObject);
var
  H: THandle;
begin
  H := ExecuteFile('Readme.exe', 'Readme.txt', '.',
          SW_SHOWNORMAL);
           if Integer(H) < 32 then
      ShowMessage('Error running Readme.exe');
end;


end.
```

يوضح OnClick الخاص بزر الـ Click Me كيفية تشغيل برنامج من داخل آخر. فى النسخ السابقة من Delphi، كانت الطريقة الصحيحة لفعل هذا هى باستدعاء الـ WinExec function. هذا مازال يعمل، ولكنه نادر. فى ظل الـ Windows 95، و الـ Windows 98، والـ Windows NT ذو الـ ٣٢ بت، من المفضل ان تستدعى التطبيق الـ Windows API ShellExecute function.

ويتطلب فعل هذا مزيداً من العمل أكثر مما تحتاج غالباً. لحسن الحظ، هناك حل ايسر، كما هو موضح فى القائمة. تحتوى ادلة Doc و Demos التابعة لـ Delphi تحتوى على unit منفصلة، وهى FMXUtils.pas، فى مجلد الـFilemanex. اضف هذه unit الى المشروع باستخدام الـ Project|Add to project. بالإضافة الى ذلك، اضف التعريف التالى الى الـ form unit، تحت كلمة الـ implementation الخاصة بها مباشرة:

```
uses FMXUtils;
```

يمكن ان تستدعى العبارات الآن الـ ExecuteFile الخاصة بالـ unit كما موضح فى التطبيق لتشغيل برنامج آخر. يمكنك حتى تنفيذ ملفات البيانات، التى تفتحها داخل التطبيقات المتعلقة بها. على سبيل المثال، قم بتمرير ملف نص الى الـ ExecuteFile لفتح هذا الملف باستخدام التطبيق الافتراضى لنوع الملف- وهو غالباً الـWindows Notepad utility.

Caution **تحذير:** اذا كانت لديك مشكلة فى تشغيل تطبيق الـ Runme على القرص المدمج المرفق بهذا الكتاب، فإن السبب غالباً ان FMXUtils.pas الخاص Delphi غير موجود فى المكان الذى يتوقعه فيه البرنامج. استخدم الـ Windows Explorer لوضع الملف، ثم استخدم أمر الـ Project|Add to project لإضافة الـ unit للـ Runme استخدام أمر الـ Project|Remove from project لإزالة المرجع القديم الى الـ FMXUtils.pas.

يتم تعريف الـ ExecuteFile function والـ parameters الخاصة بها كما يلى:

```
function ExecuteFile(const FileName, Params,
  DefaultDir: string; ShowCmd: Integer): THandle;
```

- **FileName**: يمرر اسم ملف string أو اسم مسار تام في هذا الـ parameter.
- **Params**: يمرر في هذا الـ string أية parameters، خيارات، أو اسماء ملف تريد تمريرها الى ملف الـ code في الـ FileName. يقوم برنامج العرض بتمرير الـ Readme.txtفي هذا الـparameter.
- **DefaultDir**: حدد هذا الـ string بالدليل الذى يبحث فيه الـ ExecuteFile عن الـ FileName. لاحظ اذا كان الـ Params هو اسم ملف آخر، فإن هذا التحديد لن يجعل البرنامج المنفذ يبحث في الدليل المحدد.
- **ShowCmd**: هذا يمكن ان يكون أى ثابت من ثوابت الـ SW_SHOW....مثل الـ SW_SHOWNORMAL لعرض نافذة تطبيق في حالة البدء الطبيعية لها، أو قيمة أخرى مثل الـ SW_SHOWMAXIMIZED لفتح النافذة على الشاشة الكاملة عند بدء البرنامج. انظر ملف الـ Win32.hlp لثوابت الـ SW_SHOWالاخرى التى يمكنك تمريرها للـ ExecuteFile.

مثل الـ WinExec، يقوم الـ ExecuteFile (و الـ Windows API ShellExecute function)، اذا كان ناجحاً، بادخال قيمة عدد صحيح أكبر من أو تساوى ٣٢ وأى قيمة اقل من الـ ٣٢ تشير الى وجود خطأ والقيمة المدخلة هى بالفعل الـ handle الحالى للتطبيق المستهدف، بالرغم من إن هذه الحقيقة ليست بالغة الأهمية. ان الـ WinExec البائدة ادخلت هذه القيمة كعدد صحيح، مما جعلها اسهل فى الاستخدام بهذا الحال. و الـ ExecuteFile الجديدة (و الـ ShellExecute) تدخل القيمة الناتجة كـ handle. ان فحص قيمة العدد الصحيح الخاص بها يتطلب استخدام تعبير الـ type-cast مثل :

```
.if Integer(H) < 32 then
  ShowMessage('Error running Readme.exe');
```

واخـــيـــراً، قـم بتـــجـــربة التـــالى . اضف الـ FMXUtils.pas unit للمشروع، واضف uses FMXUtils تحت كلمة implementation مباشرة، ثم أضف Button object على form قم ببرمجة الـ Button كما يلى :

```
procedure TForm1.Button1Click(Sender: TObject);
var
    H: THandle;
begin
  H := ExecuteFile('C:\Windows\Sol', '', '', SW_SHOWNORMAL);
end;
```

يتم تمرير strings (empty) null للـ string parameters الغـــيـــر مستخدمة . اضغط F9 لتشغيل البرنامج، ثم اضغط الزر لتنفيذ لعبة الـ Solitaire الخاصة بالـ Windows .

فـيـمـا يلى الـ code العـائدة ذات الـ 16-bit التى تقـوم بنفس الشئ فى الـ Windows 3.1 . وهذه مازالت تعمل، ولكن يجب استبدالها بالبرمجة الجديدة الموجودة فى المثال السابق :

```
{ This is the obsolete 16-bit method.
  Don't use this technique any longer. }
procedure TForm1.Button1Click(Sender: TObject);
var
  K: Integer;
begin
  K := WinExec('C:\Windows\Sol', SW_SHOWMAXIMIZED);
end;
```

إدارة النص فى الـ Memo objects الخاصة بك:

من الطبيعى أن تحتاج الى إضافة، حذف، إدخال وتعديل النص فى الـ Memo الخاصة بك . إن برمجة هذه المهام ومهام أخرى يتطلب فهم جيد لكيف توفر الـ Memo الوصول الى سطور النص الخاص بهم . يمكنك الوصول الى نص الـ Memo :

* كـأنه قـيـمـة الـ string لخـاصـيـة الـ Text، والتى لا تظهـر فى الـ Object Inspector وتعتبر متوفرة فقط فى وقت التشغيل . استخدم هذه الخاصـيـة فقط

عندما تريد استغلال نص الـ Memo كأنه objects . لم تعد قيمة هذه الخاصية محددة بـ ٢٥٥ رمزاً كما كانت فى الأصل فى النسخة الأولى من Delphi .

* كقاعدة من strings فى خاصية الـ Lines، باستخدام الـ methods الموضحة فى الباب التاسع لـ TStrings objects والـ TStringList . بشكل عام، تعبير هذه هى أكثر الطرق تنوعاً للتوصل الى نص الـ Memo وهى دائماً الأفضل عندما تحتاج أن تعالج سطور النص بمفردك .

لعرض واستخدام نص الـ Memo على أنه string فردى، استخدم خاصية الـ Text . وهى تعتبر موروثة من الـ TControl ولكنها غير مذكورة فى نافذة الـ Object Inspector . على سبيل المثال، أضف Button و Memo على from، واستخدم هذه العبارة فى OnClick الخاص بالـ Button لعرض نص الـ Memo :

```
ShowMessage(Memo1.Text);
```

إلصق بعض النص فى نافذة الـ Memo1 واضغط الزر . وكأن هذا يستخدم لإزالة محتويات الـ Memo ليكون الحد الأقصى لها ٢٥٥ رمزاً، ولكن الآن تستخدم الـ Delphis components تلك، نوع الـ String datas الديناميكى، ولم تعد هذه المشكلة تحدث .

للتوصل الى سطور فردية فى الـ Memo، استخدم خاصية الـ Lines . ولأن هذا يعد object من نوع الـ TStrings ، يمكنك نقل النص بين objects أخرى لها خصائص TStrings . على سبيل المثال، استخدم عبارة مثل التالية لنسخ الـ Items الخاصة بالـ ListBox للـ Lines التابعة للـ Memo :

```
Memo1.Lines := ListBox1.Items;
```

قد تستخدم التقنية العكسية لتمكن المستخدمين من إدخال بنود الـ ListBox باستخدام خصائص text-editing فى الـ Memo . قم بأداء التعيين فى الإتجاه الآخر، من الممكن أن يكون فى OnClick الخاص بالـ Button :

```
ListBox1.Items := Memo1.Lines;
```

لإدخال نص جديد فى الـ Memo، يمكنك تعيين strings لخاصية الـ Text . ولكن كن على حذر من هذه الطريقة- فهى تحل محل محتويات الـ Memo بأكملها، بغض النظر عن الحجم :

```
Memo1.Text := 'I''d rather be sailing!';
```

إن علامتى التنصيص الفرديتين تنشئ علامة الفصلة العلوية الدالة على حذف حرف (') داخل الـ string . بالرغم من أن العبارة السابقة تعمل بشكل جيد، ولكن من الأفضل أن تستخدم خاصية الـ Lines. هذه العبارة تضيف string لقائمة الـ Lines:

```
Memo1.Lines.Add('Bananas are not Oranges');
```

امسح محتويات الـ Memo باستدعاء الـ Clear method كهذا :

```
Memo1.Lines.Clear;
```

ولكن لأن الـ Memo له الـ Clear method الخاص به، يمكنك استدعاء الـ Clear دون الإشارة الى الـ Lines. هذه العبارة مساوية للعبارة السابقة :

```
Memo1.Clear;
```

حدد اذا ما كان الـ Memo خالياً بفحص خاصية الـ Count للـ Lines :

```
if Memo1.Lines.Count = 0 then
  ShowMessage('You just clicked this button!')
else
  Memo1.Clear;
```

الوصول الى الـ Memo text buffer:

كما ذكرت، يمكنك، فى أغلب الحالات، التوصل الى نص الـ Memo بسهولة من خلال الـ Lines object الخاص به. ولكن، اذا كنت تحتاج أن تؤدى عمليات مباشرة على text buffers ، يمكنك التوصل لنص الـ Memo على أنه array متجه من الـ Char . قد يكون هذا مفيداً فى بعض التطبيقات، بالرغم من أنه مع نوع بيانات الـ String الغير محددة الطول للـ Object Pascal، لم تعد التقنيات ذات قيمة عالية كما كانت فى الماضى. رغم ذلك، فإن هذا الفصل يوضح الأسس. حتى اذا لم تكن تستخدم هذه الطرق، يمكن أن يساعدك هذا الفصل فى تحديث التطبيقات القديمة التى تستخدم null-terminated strings مع الـ Memo objects.

هناك خمسة methods للعمل مع نص الـ Memo على أنه الـ Char array - أى أنه null-terminated strings . والـ methods بالترتيب الذى أوردته هو :

- **SetTextBuf:** ينسخ النص فى الـ Memo وهذا مسار لرسالة الـ wm_SetText .
- **SetSelTextBuf:** يحل النص المختار فى الـ Memo بنص جديد وهذا مساو لرسالة الـ em_ReplaceSel message .
- **GetTextLen:** يحدد عدد الرموز فى الـ Memo buffer وهذه القيمة لا تشمل الإلغاء الذى ينهى الـ buffer وهذا مساو لرسالة الـ wm_GetTextLength .
- **GetTextBuf:** ينسخ نص الـ Memo الى Char array لبرنامج أو الـ PChar buffer وهو مساو لرسالة الـ wm_GetText .
- **GetSelTextBuf:** ينسخ النص المختار من الـ Memo الى برنامج Char buffer array أو الـ PChar buffer مساو لرسالة الـ em_GetSel يتبعها نسخة string للنص المختار بالـ Windows .

يمكنك استخدام هذه الـ methods مع arrays من الـ Char Bytes ، أو مع الـ PChar pointers . يمكنك تمرير Char array الى الـ PChar . على سبيل المثال ، مع هذا التعريف التالى ، يمكنك تمرير إما Sأو P الى أى PChar parameter:

```
var
    S: array[0 .. 128] of Char;
    P: PChar;
```

هذه العبارات تنسخ الـ String sالمدعو P الى الـ Memo text buffer (لقد إفترضت أنك قد بدأت الـ strings فى مكان آخر) .

```
Memo1.SetTextBuf(S);  { Copy array S to Memo1 }
Memo1.SetTextBuf(P);  { Copy string at P to Memo1 }
```

فى الشرح التالى إننى استخدم الـPChar pointers لانها أكثر فعالية، خاصة للـ text buffer الكبيـر. لكن، نفس هذه التـقنيـات تعـمـل مع الـ Char array. توضح القائمة (١٠-٣) كيفية استخدام الـ SetTextBuf لنسخ null-terminated stringفى الـ Memo. (فى حـالة اذا اردت ان تجرب الـ code، فهى فى OnClick لـ Button). اولاً، يخـصص الـ StrAlloc ذاكـرة لـ string ذى ١٢٨ رمـزاً. ثم، ينسخ الـ StrCopy النص الحـرفى الـذاكـرة. وتقـوم التـعـبـيـرات #13#10 بإدخـال control codes على سطور جديدة فى نهايات للاسطر الثـلاثة الأولى. ينسـخ الـ SetTextBuf الـ string فى الـ Memo1. بعــد ذلك، يحــذف الـ StrDispose الذاكرة المخصصة.

القائمة (١٠-٣): يوضح OnClick هذا كيفية إدخال فى Memo

```
procedure TForm1.Button1Click(Sender: TObject);
var
  P: PChar;
begin
  P := StrAlloc(128);
     try
       StrCopy(P,
          'Red skies at night,'#13#10 +
            ' Sailor''s delight;'#13#10 +
            'Red skies at morning,'#13#10 +
            ' Sailor take warning.');
           Memo1.SetTextBuf(P);
        finally
       StrDispose(P);
         end;
end;
```

بالرغم من اننى اريد ان اوضح هنا كـيـفـيـة اسـتـخـدام الـPChar و text buffers، من المهم ان تفهم انه مع نوع بيانات الـ String للـ Object Pascal، لم تعد البرمجة كتلك الموجودة فى القائمة (١٠-٣) ضرورية.

يمكن استبدال الـ procedure السابق بما يلى ، والذى يعد اسهل فى البرمجة ويعمل بشكل رائع . وبالإضافة الى ذلك ، فهو ايضاً أكثر أماناً لأنه لا يوجد احتمال نسيان التخلص من الـ text buffer الذى تم إنشاءه باستدعاء الـ StrAlloc :

```
procedure TForm1.Button1Click(Sender: TObject);
var
  S: String;
begin
  S :=
      'Red skies at night,'#13#10 +
        ' Sailor''s delight;'#13#10 +
        'Red skies at morning,'#13#10 +
        ' Sailor take warning.';
    Memo1.Text := S;
end;
```

تتحكم الـ Memo (وكذلك الـ Edit والـ text components الأخرى) فى ذاكرة تخزين الـ string الخاصة بهم . على سبيل المثال ، عندما تستدعى الـ SetText ، ينسخ Memo بيانات الـ string فى الذاكرة الخاصة به . وإنها لمسئوليتك ان تحذف الـ string ا الديناميكية التى تقوم بإنشاءها ، حتى بعد ان تعينها للـMemo .

استدع الـ SetSelTextBuf ليحل محل النص المختار (البارز) فى الـ Memo . على سبيل المثال ، قم بتشغيل برنامج اختيارى مع الـ event handler الموجود فى القائمة السابقة ، ثم إدخل OnClick procedure آخر للـ Button بالـ code الموجودة فى القائمة (١٠-٤) لتحل محل النص المختار برموز جديدة فى الـ Memo باستدعاء الـ SetSelTextBuf اختر بعض أو كل النص فى الـ Memo ، واضغط الزر الثانى لتستبدل النص بالنص الجديد .

القائمة (١٠-٤): OnClick لاستبدال النص المختار برموز جديدة فى الـ Memo

```
procedure TForm1.Button2Click(Sender: TObject);
var
  P: PChar;
```

```
begin
  P := StrAlloc(128);
    try
      StrCopy(P,
         'How happy is the sailor''s life,'#13#10 +
            'From coast to coast to roam;'#13#10 +
            'In every port he finds a wife,'#13#10 +
            'In every land a home.');
        Memo1.SetSelTextBuf(P);
    finally
      StrDispose(P);
      end;
end;
```

اذا لم يكن هناك نصاً مختاراً في الـ Memo، يقوم الـ SetSelTextBuf بإدخال النص الجديد في الموضع الحالي للـ cursor . لاستبدال كل النص تماماً في الـ Memo، استدع الـ SelectAll methods قبل استدعاء الـ SetSelTextBuf .

لنسخ النص من الـ Memo الى الـ Char array، استدع الـ GetTextBuf أو الـ GetSelTextBuf حدد الحجم المطلوب للـ buffer باستدعاء الـ GetTextLen، قم بتخصيص كل هذه الذاكرة زائد بايت واحد لـ null terminator، و استدع الـ GetTextBuf لنسخ نص الـ Memo في المساحة المخصصة. استدع الـ GetSelTextBuf لنسخ النص المختار فقط الى الـ buffer .

توضح القائمة (١٠-٥) كيفية استخدام الـ GetTextLen و الـ GetTextBuf . يحدد الـ procedure أولاً طول الـ buffer الخاص بالـ Memo1 . اذا كان هذا الطول اكبر من صفر، يستدعي الـ code الـ StrAlloc، الذى يخصص Len+ بايت واحدة ويعين عنوان الذاكرة بـ P . ينسخ الـ GetTextBuf نص الـ Memo1 الى الذاكرة المخصصة، تأتى بعدها for loop- تبدل كل حرف في الـ buffer بالـ FF العشرية . ينسخ الـ SetTextBuf الـ Memo مرة اخرى في الـ object. واخيراً، يقوم الـ StrDispose بحذف الذاكرة

المخصصة . جرب هذا الـ procedure واضغط الزر لربط النص من والى الـ form .
ينسخ الـ procedure نص الـ Memo1 الى حاجز ديناميكى ، يبدل كل حرف يتبع لعملية لـ exclusive-or ، ثم ينسخ الـ buffer مرة اخرى الى الـ Memo .

القائمة (١٠-٥): استخدم OnClick للتعامل مع memo object text

```
procedure TForm1.Button3Click(Sender: TObject);
var
 I, Len: Integer;
    P: PChar;
begin
    Len := Memo1.GetTextLen;
    if Len = 0 then
     ShowMessage('Text buffer is empty!')
       else begin
     P := StrAlloc(Len + 1);
        try
          Memo1.GetTextBuf(P, Len);
             for I := 0 to Len do
               P[I] := Chr(Ord(P[I]) xor $ff);
                Memo1.SetTextBuf(P);
            finally
          StrDispose(P);
            end;
       end;
end;
```

توضح القائمة جانباً من العمل مع الـ PChar pointers . بالرغم من ان متغير PChar يعتبر pointer الى الـ Char byte ، يمكنك استخدامه وفكر به كأنه array .
على سبيل المثال ، تعين هذه العبارة الحرف A للرمز الأول string buffer المسمى بـ P:

```
P[0] := 'A';
```

فى القائمة (١٠-٥) يمكنك استبدال الـ GetTextBuf بالـ GetSelTextBuf لنسخ الخاص بك . فيما عدا هذا تعتبر الـ functions شئ

واحداً. (اذا جربت هذا، إختر كل النص كل مرة قبل ضغط الزر، أو يكون النص لم يتم التعامل بشكل سليم).

فهم الـ Tab والـ Enter:

اعتماداً على احتياجات واجهة التطبيق الخاصة بك للـMemo، فإن ضغط الـ Enter و الـ Tab يمكن ان يكون له تأثيرات متنوعة. على سبيل المثال، قد تريد ان يقوم المستخدمون بإدخال سطور متعددة في نافذة الـ Memo- في هذه الحالة، فإنك في الغالب تريد ضغط الـ Enter لتبدأ سطر جديد. اذا لم تكن تريد ان يعمل الـ Enter بهذه الطريقة، يمكنك برمجة المفتاح على ان يحول الـfocus الى control اخر. يمكنك ايضاً تحديد ما اذا كان ضغط مفتاح الـ Tab يؤدي إلى إضافة الـ tab control code في نص Memo، أو يحول الى الـ control التالي في ترتيب الـ Tab.

حدد خاصية الـ WantReturns بـ True لإدخال سطور جديدةفي نص الـ Memo عندما يضغط المستخدمون الـ Enter. حدد خاصية الـ WantReturns بـ False لتوجيه ضغط الـ enter الى الـ form. ولكن، هذا التحديد يتطلب تحديداً مناسباً لخاصية الـ KeyPreview للـ form. وعادة، اذا كانت الـ WantReturns محددة بـ True ، يجب ان تكون الـ KeyPreview محددة بـ False حتى يتلقى الـ control، وليس الـ form ضغط مفتاح الـ Enter. اذا كان الـ WantReturns محددة بـ False، يجب ان تكون الـ KeyPreviewمحددة بـ True ؛ والا يستمر الـ form فى حالة عدم تلقى لضغط مفتاح الـ Enter فى OnKeypress الخاص به.

حدد الـ WantReturns بـ True لإدخال tab control codes فى نص الـ Memo عندما يضغط المستخدمون الـ Tab. حدد الـ WantReturns بـ False، لنقل الـ focus إلى control آخر عند تلقى ضغط مفتاح الـTab. عندما تكون الـ WantReturns محددة بـ True ، يمكنك فحص الـ Memo بضغط الـ Tab، ولكن لا يمكنك ضغط الـ Tab لتخرج.

ان ضغط Ctrl+Enter يبدأ دائماً سطر جديد فى الـ Memo، بغض النظر عن قيمةالـ WantReturns . وكذلك، فإن ضغط Ctrl+Tab تضيف دائماً tab control code بغض النظر عن قيمة الـ WantTabs.

بارسـال رسـالة Windows الى الـ Memo، يمكنك تغـيـير مسـافـات tab controls . توضح الـ code التالية هذه التقنية ويمكنك استخدامها فى OnCreate الخاص بالـ form . ان الـ Tabstops typed-constant array له قيمة Integers واحدة فقط- فى هذه الحالة، محددة بـ ١٨ . ولان اغلب الـ fonts نسبية، فإن الـ Windows يقـيس عـدد مـرات توقف الـ Tab فى dialog box units بدلاً من الحروف، (ان تحـديد الـ Tab الافتـراضى هو ٣٢ و dialog units). ويستـدعى OnCreate الـ Perform method (كل الـ components لديهـــا هذا الـ methods) لإرسـال رسـالة em_SetTabStops للـ Memo. ان الـ parameter الواحـد هو الذى يحـدد للـ TabStops مـدخل واحـد، يتكرر لكل tabs . انظر الـ em_SetTabStops فى ملف الـ Win32.hlp لمزيد من المعلومـات حـول تحـديد مواصفات الـ tabs فى .text controls .

والـ code التـالى ايضـاً يوضح كـيـفـيـة تمرير مـتـغـيـر الى الـ LParam parameter الخاص بالرسالة وذلك بطريقة. الـ type-casting بالتحويل الى قيمة الـ Longint . فى هذا المثـال، تحـدد عـلامـة الـ @ عنوان الـ TabStops array . والتعـبيـر التام Longint(@TabStops) يمرر عنوان الـ array كقيـمـة Longint ذات ٣٢ بت :

```
procedure TForm1.FormCreate(Sender: TObject);
const
  TabStops: array[0 .. 0] of Integer = (18);
begin
  Memo1.Perform(em_SetTabStops, 1, Longint(@TabStops));
end;
```

لتـمرير أكـثـر من tab stop واحـدة، قم بتغيير تعريف الثـابت فى الـ code السابقة لشئ مثل :

```
const
  TabStops: array[0 .. 3] of Integer = (4, 8, 12, 16);
```

وكـذلك قـم بتـغـيـيـر الرقم واحـدة الى رقم اربع فى اسـتـدعـاء الـ Memo1.Perform . اذا كنت تمرر tab stop واحدة فـقط كـمـا فى القـائمـة، لا

يجب عليك إنشاء array. يمكنك بدلاً من ذلك تعريف الـ TabStops على انه عدد صحيح (والتى، بعد كل شئ، هى نفسها كالـ array التى لها قيمة واحدة.

الـText و Clipboard:

بالرغم من نفعيتها المحدودة، تعتبر الـ Clipboard الخاصة بالـ Windows محطة مشهورة لبث المعلومات من والى التطبيق. ان غالبية مستخدمى الـ Windows يتعلمون بسرعة كيف يقصون وينسخون النص، بالرغم من انهم قد لا يدركون انهم يستخدمون الـ Clipboard للقيام بهذه الاعمال. ولان Edit components، الـ MaskEdit، والـ Memo تدعم تلقائياً القص، والنسخ، واللصق، فإن استخدام هذه الـ objects هو كل ما تحتاجه لتوفر خدمات الـ clipboard فى تطبيقك.

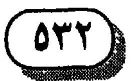
ملحوظة: والـ Database components مثل الـ TDBEdit، TDBMaskEdit، TDBImage، والـ TDBMemo أيضاً تدعم الـ clipboard methods الموضحة هنا.

لكى تجعل clipboard أسهل استخداماً، وخاصة للمبتدئين الذين لم يتمكنوا من مهارات أوامر لوحة مفاتيح الـ Windows، استدع الـ methods CopyToClipboard، الـ CutToClipboard، والـ PasteFromClipboard فى الـ Edit، الـ MaskEdit، والـ Memo. على سبيل المثال، لقص النص المختار من الـ Memo الى الـ clipboard، استخدم هذه العبارة، ربما فى OnClick الخاص بالـ Button، أو فى الـ procedure الخاص بأمر الـ Edit|Cut الخاص بالقائمة:

```
Memo1.CutToClipboard;
```

وتعمل الـ procedure الأخرى بطريقة مشابهة ولا تتطلب أية parameters. والطريق الى النجاح هو أن تدرك أن الـ CopyToClipboard والـ CutToClipboard لا يعملان إلا على النص المختار فقط. لنسخ كل نص الـ Memo على الـ clipboard، إستدعى الـ Selectall قبل تنفيذ هذه العملية:

```
Memo1.SelectAll;
Memo1.CopyToClipboard;
```



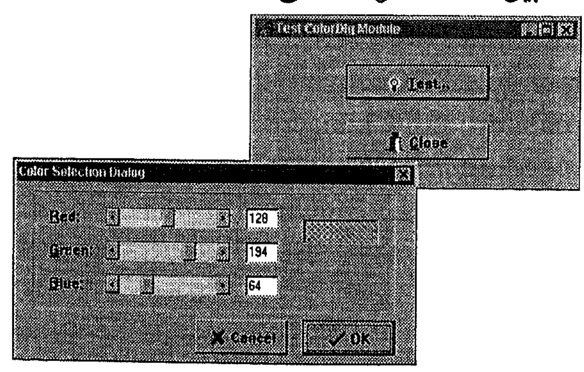
توفر العديد من الـ components إمكانات الـ scrolling تلقائياً ولا تتطلب أية برمجة. على سبيل المثال، الـ Memo يعرض، scroll bar رأسية وافقية على حسب الحاجة لتستطيع الوصول الى النص. ويعمل الـ Scrolling كما هو متوقع بالضبط.

بالنسبة لمهام الـ scroll الأخرى، لديك اختياران. يمكنك إضافة ScrollBar على form أو يمكنك استخدام ScrollBar لتوفير مسطح قابل للـ scrolling يمكن أن يحتوى على objects أخرى. يوضح الفصلان التاليان كيفية الـ Scrolling فى مع هذين الـ components الهامتين النافعتين.

التحريك بالـ ScrollBar:

On the CD-ROM

على القرص المدمج: يقوم الـ ScrollBar Component بإنشاء scrollBar object مستقل يمكنك إضافته الى form أو أية حاوية أخرى. بكتابة event handler واحد، يمكنك إنشاء code تستجيب للتغييرات فى موضع الـ ScrollBar. يمكن أن يكون لديك أى عدد تحتاجه من الـ ScrollBar– فهى وسائل بيئة جيدة فى اختيار قيم فيما بين مدى منخفض أو مرتفع. كمثال على هذا المفهوم، قم بتجربة تطبيق الـ Test فى دليل الـ Source\ColorDlg على القرص المدمج. قم بتشغيل تطبيق الـ Test فى دليل الـ ColorDlg، واضغط زر الـ Test لعرض dialog اختيار اللون الموضح هنا. لاختيار قيم الأزرق، والأخضر، والأحمر للألوان الموضحة فى شكل (١٠-٣) الـ source code لتطبيق الـ Test موضحة فى القائمة (١٠-٦).



شكل (١٠-٣): الـ Color Selection Dialog يظهر كيفية استخدام الـ ScrollBar

قائمة (١٠-٦): \ColorDlg\ColorDlg.pas

```
unit Colordlg;

interface

uses
  Windows, SysUtils, Messages, Classes, Graphics,
    Controls, Forms, Dialogs, StdCtrls, Buttons, ExtCtrls;

type
    TColorDlgForm = class(TForm)
      RedSB: TScrollBar;
         GreenSB: TScrollBar;
         BlueSB: TScrollBar;
         RedLabel: TLabel;
         GreenLabel: TLabel;
         BlueLabel: TLabel;
         RedEdit: TEdit;
         GreenEdit: TEdit;
         BlueEdit: TEdit;
         ColorEdit: TEdit;
         OkBitBtn: TBitBtn;
         CancelBitBtn: TBitBtn;
         Bevel1: TBevel;
         procedure FormCreate(Sender: TObject);
         procedure SBChange(Sender: TObject);
         procedure EditChange(Sender: TObject);
         procedure FormActivate(Sender: TObject);
         procedure OkBitBtnClick(Sender: TObject);
         procedure CancelBitBtnClick(Sender: TObject);
       private
      RedPos, GreenPos, BluePos: Integer;  { For undo }
         EditControls: array[0 .. 2] of TEdit;
```

```
      ScrollBars: array[0 .. 2] of TScrollBar;
        procedure UpdateColor;
       public
      ColorResult: TColor;  { Selected color }
       end;

var
 ColorDlgForm: TColorDlgForm;

implementation

{$R *.DFM}

{- Update ColorResult using scrollbar positions }
procedure TColorDlgForm.UpdateColor;
begin
    ColorResult := RGB(
     RedSB.Position, GreenSB.Position, BlueSB.Position);
      ColorEdit.Color := ColorResult;  { Show color }
end;

{- Initialize TObject control arrays }
procedure TColorDlgForm.FormCreate(Sender: TObject);
begin
 EditControls[0] := RedEdit;
    EditControls[1] := GreenEdit;
    EditControls[2] := BlueEdit;
    ScrollBars[0] := RedSB;
    ScrollBars[1] := GreenSB;
    ScrollBars[2] := BlueSB;
end;

{- Update values in Edit boxes for ScrollBar changes }
```

```
procedure TColorDlgForm.SBChange(Sender: TObject);
begin
  with Sender as TScrollBar do
      EditControls[Tag].Text := IntToStr(Position);
        UpdateColor;
end;

{- Update scrollbar positions for Edit box changes }
procedure TColorDlgForm.EditChange(Sender: TObject);
begin
    with Sender as TEdit do
      ScrollBars[Tag].Position := StrToInt(Text);
end;

{- Save scrollbar positions for possible undo }
procedure TColorDlgForm.FormActivate(Sender: TObject);
begin
  RedPos := RedSB.Position;
     GreenPos := GreenSB.Position;
     BluePos := BlueSB.Position;
end;

{- Respond to OK button. Accept changes. }
procedure TColorDlgForm.OkBitBtnClick(Sender: TObject);
begin
  ModalResult := mrOk; { Close Window. Color in ColorResult. }
end;

{- Respond to Cancel button. Undo changes. }
procedure TColorDlgForm.CancelBitBtnClick(Sender: TObject);
begin
  RedSB.Position := RedPos;
     GreenSB.Position := GreenPos;
```

```
    BlueSB.Position := BluePos;
     ModalResult := mrCancel;
  end;

  end.
```

يذكر الجدول (١٠-٢) بعض الخصائص الرائعة للـ ColorDlg components، والتى توضح بعض التقنيات البيئية الممتعة. تحدد الـ labels الثلاث الـ Alt accelerator فى خصائص الـ Caption التابعة لهم، وتحدد قيم الـ FocusControl لهم للـ Edit المناسب. على سبيل المثال، قم بتشغيل برنامج الـ Test واضغط Alt+G لاختيار الـ GreenEdit.

إن كلاً من الـ ScrollBars الثلاثة فى النافذة لها غالباً قيمة LargeChange. هذا يؤثر على كم الـ scrolling عندما يضغط المستخدمون داخل الـ bar– فى أغلب الحالات، يجب أن تحدد الـ LargeChange أكبر من الـ SmallChange (التى تساوى واحد عادة)، والذى يؤثر على كم الـ scrolling لأزرار الأسهم للـ ScrollBars وتحدد الـ ColorDlg ScrollBars قيم الـ Max. الخاصة بهم أيضاً بـ ٢٥٥. تمثل نوع بيانات الـ TColor الخاصة بـ Delphi الألوان لمجموعة من الأحمر والأخضر والأزرق (RGB)، كلاً منها فى المدى من صفر الى ٢٥٥.

جدول (١٠-٢): خصائص الـ ColorDlg

Component	*Name*	*Property*	*Value*
Form	ColorDlgForm	Caption	Color Selection Dialog
Label	RedLabel	Caption	&Red:
		FocusControl	RedEdit
Label	GreenLabel	Caption	&Green:
		FocusControl	GreenEdit
Label	BlueLabel	Caption	&Blue:
		FocusControl	BlueEdit
ScrollBar	RedSB	LargeChange	10
		Max	255
		Tag	0

ScrollBar	GreenSB	Tag	1
ScrollBar	BlueSB	Tag	2
Edit	RedEdit	Tag	0
Edit	GreenEdit	Tag	1
Edit	BlueEdit	Tag	2

لاحظ أيضاً فى جدول (١٠-٢) أن الـ Edit والـ ScrollBar تحدد قيم الـ Tag الخاصة بها بصفر، وواحد، وإثنين. يستخدم البرنامج هذه القيم لربط الـ components فى وقت التشغيل. لترى كيف يعمل، اختر الـ TColorDlgForm class فى القائمة (١٠-٦). يعرف الـ private section الـ arrays من نوع الـ objects:

```
EditControls: array[0 .. 2] of TEdit;
ScrollBars: array[0 .. 2] of TScrollBar;
```

توفر الـ arrays وسائل يسيرة للوصول الى كلاً من الـ forms أو الـ Edit والـ ScrollBar. تذكر أن متغيرات الـ TEdit والـ TScrollBar تعد references. لذا فالـ arrays يحتويان على pointers الى الـ objects، وليست الـ objects ذاتها. لبدء الـ array، يقوم OnCreate الخاص بالـ form بتعيين objects لمواضع الـ array، تساوى قيم الـ Object Tag. على سبيل المثال، تعين هذه العبارة الجزء الأول فى الـ EditControls لتشير الى الـ RedEdit:

```
EditControls[0] := RedEdit;
```

بعد إعداد component reference arrays، يمكن للبرنامج أن يستخدم قيم الـ Tag كـ index للوصول الى الـ objects الفردية. على سبيل المثال، اختبر الـ SBChange procedure، الذى يتعامل مع الـ OnChange event للـ ScrollBar. إن كل OnChange خاص بـ ScrollBar محدد بنفس الـ event handler. فى الـ procedure، يعتبر الـ Sender parameter الذى تم تمريره للـ procedure هو ScrollBar الذى قد تغيره. لتحديث الـ Edit المرتبط به بحيث تتماشى قيمة العدد الصحيح الخاص به مع موضع الـ ScrollBar، يستخدم الـ procedure عبارة with ليعامل الـ Sender على أنه TScrollBar object، ثم ينفذ هذه العبارة:

```
EditControls[Tag].Text := IntToStr(Position);
```

يقوم الـ IntToStr بتحويل قيمة خاصية الـ ScrollBar Position الرقمية إلى string، ويعين الناتج لخاصية الـ Text التابعة للـ Edit control، باستخدام الـ EditControls array والـ Tag index للإشارة الى الـ Edit المرتبط.

يؤدى الـ EditChange procedure مهمة شبيهة. عندما تدخل قيمة جديدة فى نافذة الـ Edit، فإن الـ OnChange الناتج يقوم باستدعاء الـ EditChange. يستخدم الـ procedure عبارة الـ with ليعامل الـ Sender على أنه TEdit، ثم ينفذ هذه العبارة:

```
ScrollBars[Tag].Position := StrToInt(Text);
```

تقوم الـ StrToInt Function بتحويل خاصية الـ Text لـ Edit الى قيمة عدد صحيح، والتى تحددها مرة أخرى العبارة إلى ScrollBar Position. هذا يجعل الـ thumb box فى ScrollBar يتنقل الى موضع يتلائم مع الـ Edit. إن الـ ScrollBar array وخاصية الـ Tag يجعلا من السهل اختيار الـ ScrollBar المرتبط بالـ Edit الذى ولد الـ OnChange.

إن الـ SBChange procedures و EditChange يتعاملان مع جميع الـ OnChange event للـ Edit والـ ScrollBar الستة الخاصة بالـ dialog. يمكن للـ procedures أن يتعاملا مع أي عدد من الـ objects المرتبطة، ويعرضا تقنية مفيدة فى تقليل حجم الـ code. عندما يكون لديك component object عديدة مرتبطة بالنافذة، فكر فى استخدام array و Tag index للإشارة إليهم فى الـ event handler المشتركة.

يوضح الـ Procedure UpdateColor كيفية استخدام قيم الـ ScrollBar Position. فى هذه الحالة، ترجع الـ RGB function قيمة LongInt بإعطاءها ثلاثة بايت جزئية تمثل ألوان الأزرق والأخضر والأحمر. يحدد البرنامج اللون الناتج الى متغير الـ ColorResult، المعرف فى الـ public section للـ TColorDlgForm class.

يعرض الـ UpdateColor أيضاً اختيار اللون التالى باستخدام خدعة بسيطة أسرع من الأسلوب الجرافيكى الواضح. فى إحدى نسخ البرنامج، قمت باستخدام الـ Shape لعرض اللون، ولكن بسبب الطريقة التى يعمل بها الـ Windows، يؤدى كل تغيير باللون الى أن يمسح الـ Shape نفسه ليصبح ذا لون أبيض، مما قد

ينتج وميضاً مزعجاً. (ولكن، اعتماداً على نظامك، قد لا ترى هذا). لحل المشكلة، قمت باستخدام الـ Edit، حددت الـ Enabled والـ TabStop بـ False، وحددت الـ ReadOnly بـ True. لا يمكنك اختيار أو إدخال نص فى الـ Edit الناتج. يقوم الـ UpdateColor Procedure بتعين الـ ColorResult الى خاصية الـ Color بالـ Edit، والذى لا تعيد رسم نفسها بين عمليات التحديث. عندما تحتاج مربع ملون بسيط، جرب الـ Edit بدلاً من الـ Shape.

يقوم الـ ColorDlg بتنفيذ ميزة الـ undo، والتى تحتاجها جميع الـ dialogs التى لديها زر Cancel فى هذه الحالة، يقوم OnActivate الخاص بالـ form بحفظ قيمة الـ Position للـ ScrollBar الحالى فى ثلاث متغيرات الـ RedPos، الـ GreenPohs، و الـ BluePos . اذا ضغط المستخدم الـ Close لإنهاء الـ dialog، يقوم OnClick الخاص بالزر بتعيين القيمة التى حفظها الى خصائص الـ Position للـ ScrollBar مرة اخرى.

للوهلة الأولى، قد لا تبدو هذه التعينات كافية لإلغاء كل تغييرات الـ dialog تماماً. والذى لا يبدو واضحة هو ان تعين قيم جديدة للـ ScrollBar يولد OnClick events . لذلك، فإن ثلاث تعينات فى CancelBitBtnClick تؤدى الى ثلاث استدعاءات للـ SBChange، والذى يعيد تحديد الـ Edit controls ويقوم بتحديث نافذة اللون. تتبع هذه الـ events بضغط F8 و F7 لتشغيل برنامج الـ Test، أو حدد نقاط إنطلاق فى مواضع استراتيجية، وتحقق من التتابع الذى يحدث عندما تضغط الـ Cancel.

لاستخدام dialog اختيار اللون، اضف الـ ColorDlg.pas الى أى مشروع وأضف ColorDlg للـ uses directive للـ form الرئيسية. استدع الـ ShowModal function لعرض نافذة الـ dialog:

```
ColorDlgForm.ShowModal;
```

يرجع الـ ShowModal قيمة تشير الى أى زر تم ضغطه لإنهاء الـ dialog. يمكنك فحص هذه القيمة واتخاذ التصرف المناسب. على سبيل المثال، بدلاً من إستدعاء الـ ShowModal كما فى السطر السابق، استخدم هذه الـ code المعقدة بعض الشئ:

```
with ColorDlgForm do
if ShowModal = mrOk then
  ShowMessage(Format('Color value = $%-.6x', [ColorResult]));
```

اذا أرجع الـ ShowModal الـ mrOk، يعـرض الـ ShowModal قـيـمـة الـColorResult من الـColorDlgForm ، باستخدام الـ Formatلإدخال القيمة بالكسـر العـشـرى فى dialogالرسـالة . ويقـوم النص ذو الشكل المضـحك ، وهو,"%-.6x" ، بتهيئة الـ ColorResult على انه قيمة عشرية (x) ذات مسافة (-) فى ستة اعمدة ذات اصفار من ناحية اليسار (6.) .

ان الـ forms والـ components الاخرى تستـخدم الـ TScrollBar class كخاصية للـ scroll bars الرأسى والأفقى . ان هذه الـ class لها نفس خصائص والـ methods كما للـ TScrollBar clas، التى تمكنك من استخدامها كـobjects مستقلة على الـ form .

التحريك بالـScrollBox:

ان الـ ScrollBox يشبه الـ Panel التى يمكن أن تتحرك الى أعلى ، واسفل ، يساراً ويميناً . يعتبر الـ ScrollBox حاوية يمكن ان تحمل objects اخرى الازرار ، الـ labels، الـ Edit، الـ radio buttons- وحـــتى الـ panels الـ ScrollBox الأخرى .

ان إنشاء الـ ScrollBox يعتبر أمر سهلاً فقط أضف واحداً على الـ form، وحـدد حـجـمـة أو حـدد الـ Align بـ alClient لتـمـلأ الـ panel الـ client area الخاصة بالنافذة . ويعتبر هذا التحديد جيد لإنشاء dialog boxes قابلة للتحريك وforms لإدخال البيانات ذات controls كثيرة جداً بحيث تتناسب بشكل مريح داخل نافذة واحدة . تحدث الـ Scrollingبصورة تلقائية ، وتظهر الـ scroll bars حسب الحاجة لتمكن المستخدمين من الوصول الى الـ controls الواقعة خلف حدود النافذة .

Tip

فكرة: وهناك طريقة اخـرى للـform، وهى بأن تحـدد خـاصـيـة الـ AutoScroll لهـا بـ True . ولكن لا تزال الـ ScrollBox مـفـيـدة فى ظروف عديدة ، كما يوضح هذا الفصل .

قم بتصـميم الـ ScrollBox الخاصـة بك بالسـعة الكاملة لها . على سبيل المثال ، يمكنك تكبير نافذة الـ form ، وإضافة كل الـ controls فى المكان الذى تريده فيه . بعد تصميم النافذة ، قم بتقليص الـ form المصممةلترجع الى حجمها النهائى مرة اخرى .

وتعتبر الـ ScrollBoxes مفيدة ايضاً فى حالة اختلاف قياس الـ form فى حالات الـ Resolution المختلفة . على سبيل المثال ، يمكنك تصميم full screen entry form بقياس ١٠٢٤×٦٧٨ التى لازالت تعمل بشكل صحيح على الحاسبات المعملية بقياس ٦٤٠×٤٨٠ . على الحاسب المعملى ، يجب على المستخدم تحريك النافذة ليصل الى كل الـ controls .

ان الـ controls التى يمكن ان تتلقى input focus (مثل edit controls) تتحرك تلقائياً تجاه الرؤية . حاول إضافة إثنى عشرة Edits على الـ ScrollBox . قم بتشغيل البرنامج واضغط Tab . عندما يتغير input focus يتحرك الـ Edit الحالى تلقائياً الى إتجاه الرؤية .

فى بعض الأحيان ، قد تحتاج الى عرض controls معين . على سبيل المثال ، قد يختار المستخدم أمر برنامج أو يضغط زراً . يمكنك استدعاء الـ SetFocus method الخاص بالـ object ، أو يمكنك استدعاء الـ الخاص بالـ ScrollInView procedure . على سبيل المثال ، لضمان الـ Edit7 مرئياً ، قم بتنفيذ العبارة التالية :

```
ScrollBox1.ScrollInView(Edit7);
```

Components و controls :

عندما تتعرف اكثر على حاويات مثل الـ ScrollBox ، سوف تكتشف الحاجة الى التوصل الى الـ components بطرق متنوعة . بالرغم من ان كل component objects تعتبر متوفرة بالاسم ، مثل الـ Label2 أو الـ Edit9 ، وليس من الكافى دائماً الإشارة الى الـ objects منفردة . فى form أو ScrollBox (أو Panel) ذات عشرات الازرار ، وليس من المستحب ان تكتب code مثل التالية :

```
Button1.Enabled := False;
Button2.Enabled := False;
...
Button38.Enabled := False;
```

ان الحاويات كالـ form والـ ScrollBox تقدم الـ Components array، الذى يمكنك استخدامه للوصول الى كل الـ objects المملوكه للـ object الرئيسى . الـ ComponentCount يساوى عدد تلك الـ components . يمكنك استخدام هذه الخصائص لتكتب loops مثل هذه :

```
for I := 0 to ComponentCount - 1 do
  Components[I].Enabled := False;
```

توفر الحاويات ايضاً خصائص مشابهة للـ array والـ count والـ Controls والـ ControlIndex . ولكن هذه الخصائص تساعدك للوصول الى الـ components التى تعتبر ايضاً نوافذ صغيرة. توفر خصائص الـ Controls والـ ControlIndex وصلاً الى component objects . ويذكر والـ Controls array علاقات النافذة مع component objects . والـ Controls array تحتوى على قائمة بعلامة الـ object . ولا يعتبر الـ arrays واحداً دائماً. على سبيل المثال، يملك الـ ScrollBox للـ components الخاصة به على انها child controls window . والـ form التى تملك الـ ScrollBox تملك نفس تلك الـ objects كـ components .

على القرص المدمج: هذه المفاهيم ليست اكاديمية خالصة- يمكنك استخدامها تبعاً لمصالحك فى كتابة code تؤثر على مجموعات من الـ objects . لتفهم جيداً الفرق بين الـ Controls array والـ components، قم بتشغيل تطبيق الـ ContComp (Controls و Components) الموجود على القرص المدمج فى دليل الـ Source\ContComp . يذكر البرنامج الـ components والـ controls فى اربعة قوائم للـ ScrollBox وللـ form الرئيسية. يوضح شكل (١٠-٤) البرنامج. تذكر القائمة (١٠-٧) للـ source code، التى تتضمن procedures التى يمكنك استخدامها لفحص علاقات الـ components و form control .

شكل (١٠-٤): يعرض الـ ContComp علاقات الـ components والـ controls و الـ objects مع ScrollBox، الذى يحتوى على ثلاث RadioButton controls، والـform

القائمة (١٠-٧): ContComp\Main.pas

```
unit Main;

interface

uses
    Windows, SysUtils, Messages, Classes, Graphics,
    Controls, Forms, Dialogs, StdCtrls, Buttons;

type
 TMainForm = class(TForm)
      ScrollBox1: TScrollBox;
         Label1: TLabel;
         RadioButton1: TRadioButton;
         RadioButton2: TRadioButton;
         RadioButton3: TRadioButton;
         Button1: TButton;
         Button2: TButton;
         ScrollBoxComponents: TListBox;
         ScrollBoxControls: TListBox;
         FormComponents: TListBox;
         FormControls: TListBox;
```

```
      Label2: TLabel;
        Label3: TLabel;
        Label4: TLabel;
        Label5: TLabel;
        CloseBitBtn: TBitBtn;
        procedure FormCreate(Sender: TObject);
       private
      { Private declarations }
       public
      { Public declarations }
       end;

var
 MainForm: TMainForm;

implementation

{$R *.DFM}

procedure ListControls(ListBox: TListBox;
 Control: TWinControl);
var
 I: Integer;
begin
    with Control do
    for I := 0 to ControlCount - 1 do
     ListBox.Items.Add(Controls[I].Name);
end;

procedure ListComponents(ListBox: TListBox;
 Component: TComponent);
var
 I: Integer;
```

```
begin
    with Component do
    for I := 0 to ComponentCount - 1 do
     ListBox.Items.Add(Components[I].Name);
end;

procedure TMainForm.FormCreate(Sender: TObject);
begin
  ListControls(ScrollBoxControls, ScrollBox1);
    ListComponents(ScrollBoxComponents, ScrollBox1);
    ListControls(FormControls, MainForm);
    ListComponents(FormComponents, MainForm);
end;
end.
```

لفـحص عـلاقـات الـ control و الـ components فى تطبيـقـاتك، إنسخ الـ procedure ListComponents والـ ListControls فى أى unit module. لن تريد فى الغالب ان تتضمن الـ code فى التطبيق الأخير، لذا لم اجعلها جزءاً من class. فى أى event handler (يستخدم البرنامج الـ OnCreate الخاص بالـ form) استدع كلا الـprocedure) لإدخال الـ components والـ controls فىListBoxes . قم بتمرير الـ ListBox والـ component objects على انهما arguments قد تقوم بتمرير اية object منحدرة من الـ TComponent الى الـ ListComponents و اية object منحدرة من الـ TWinControl الى الـ ListControls. ولكن، وobject components يجب ان توفر الـ Control و الـ Components arrays.

ونعـود للـ ScrollBoxes، كـمـا يوضح شكل (١٠-٤)، ان الـ ScrollBoxes لا يملك ثلاثRadioButton component objects. ولكن، الـobjects تعتبر Child ScrollBox. من هذه المعلومة، يمكنك كتابة procedure فعالاً لأداء عمليات على جميع الـ ScrollBox control. على سبيل المثال، جرب OnClick الموجود فى القائمة (١٠-٨) للـ Button الثالث فى برنامج الـ ContComp لإبطال الـ RadioButtons الخاصة بالـ ScrollBox. يتوصل الـ code لكل Button من خلال الـ ScrollBox control.

إرجع مـرة أخـرى الى شكل (١٠-٤) "أو قم بتـشـغـيل برنامج الـ ContComp) واختـبـر قـوائم الـ FormComponents والـ FormControls . تمتلك الـ form كل الـobjects الموجـودة فى البـرنامج، بالرغم من ان الـRadioButtons الثـلاثة داخل الـ ScrollBox . ولكن، الـ control مـثل الـ Child Window يمكن ان يكون له أب واحـد فـقط، بذلك، فـإن الـ Controls array الخاص بالـ form لا يذكر الـ RadioButtons .

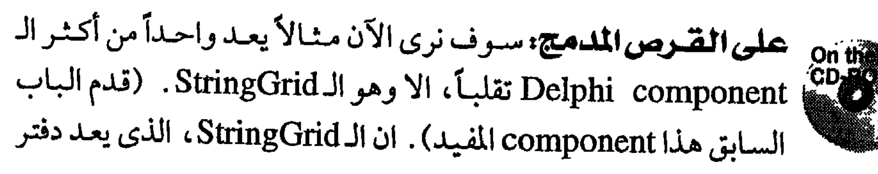
القائمة (١٠-٨): OnClick لإضافة زر

```
procedure TMainForm.Button3Click(Sender: TObject);
var
  I: Integer;
begin
  with ScrollBox1 do
    for I := 0 to ControlCount - 1 do
      Controls[I].Enabled := False;
end;
```

تتساوى قيمة خاصية الـ ComponentIndex مع قيمة الـ object index فى الـ Components array التـابعـة لـ object مـعين. بنفس الطريقـة، تسـاوى الـ Control Index الـ object index فى الـ Components array التابع لـ object مـعين. اذا كـانت الـ T Label object مملوكـة للـforms، واذا كـانت الـ T.ComponentIndex تساوى K إذن الـ Form.Components[K] تشير الى الـ T. بنفس الطريقة، اذا كانت الـ T control وهو Child Window من الـ form، فالـ Form.Controls[K] تشير الى الـ T. اذا قضيت بعض الوقت فى فحص هذه العلاقات، سوف تكتشف العديد من الفرص للوصول الى الـ object بطريقة فعالة من خلال الـ Controls array والـ Components .

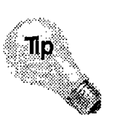
الـStringGrids:



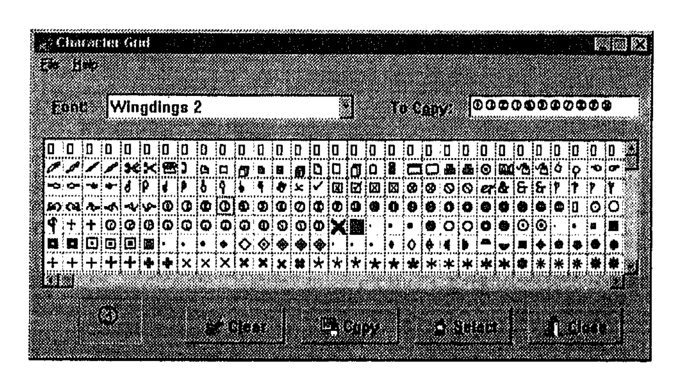


على القرص المدمج: سوف نرى الآن مثالاً يعد واحداً من أكثر الـ Delphi component تقلباً، الا وهو الـ StringGrid. (قدم الباب السابق هذا component المفيد). ان الـ StringGrid، الذى يعد دفتر

حـسـابات (أو spreadsheet) تام، ينظم الـ string و object data الأخـرى فى تخطيط جدول من صفوف واعمدة. يوضح برنامج الـ CharGrid على القرص المدمج فى دليل Source\CharGrid كيفية استخدام الـ StringGrid. كما هو موضح فى شكل (١٠-٥)، يعرض البرنامج StringGrid يوضح كل الرموز فى أى Windows font. اختر font من الـ Font ComboBox. اضغط مرتين أى رمز فى الـ Grid لإضافته الى نافذة الـ To Copy Edit. أو، يمكنك ايضاً ضغط زر الـ Select، إضغط زر الـ Copy لنسخ الحروف المختارة بالـ clipboard. يمكنك عندئذ الانتقال الى تطبيق آخر ولصق الحروف فى أى Text edit. توضح القائمة (١٠-٩) الـ source code للـCharGrid.



فكرة: استخدام الـ StringGrid. لإضافة حروف صعبة الكتابة مثل حروف حق الطبع (C) والعلامة التجارية (R) فى الـ Pascal string.



شكل (١٠-٥): تطبيق الـ StringGrid يوضح كيفية استخدام الـ StringGrid الخاص بـ Delphi

القائمة (١٠-٩): CharGrid\Main.pas

```
unit Main;

interface

uses
  Windows, SysUtils, Messages, Classes, Graphics, Controls,
    Forms, Dialogs, Grids, StdCtrls, Buttons, ExtCtrls,
```

```
    Menus,
     About;

  type
    TMainForm = class(TForm)
     StringGrid1: TStringGrid;
        FontCB: TComboBox;
        FontLabel: TLabel;
        CloseBitBtn: TBitBtn;
        CopyEdit: TEdit;
        CopyLabel: TLabel;
        SelectBitBtn: TBitBtn;
        ClipBitBtn: TBitBtn;
        CharLabel: TLabel;
        Bevel1: TBevel;
        ClearBitBtn: TBitBtn;
        MainMenu1: TMainMenu;
        FileMenu: TMenuItem;
        FileExit: TMenuItem;
        HelpMenu: TMenuItem;
        HelpAbout: TMenuItem;
       procedure FormCreate(Sender: TObject);
        procedure FontCBChange(Sender: TObject);
        procedure FontCBKeyDown(Sender: TObject; var Key:
     Word;
      Shift: TShiftState);
          procedure StringGrid1DblClick(Sender: TObject);
        procedure StringGrid1KeyDown(Sender: TObject;
         var Key: Word; Shift: TShiftState);
          procedure StringGrid1SelectCell(Sender: TObject;
         Col, Row: Longint; var CanSelect: Boolean);
          procedure FileExitClick(Sender: TObject);
        procedure HelpAboutClick(Sender: TObject);
        procedure SelectBitBtnClick(Sender: TObject);
```

```
      procedure ClipBitBtnClick(Sender: TObject);
        procedure ClearBitBtnClick(Sender: TObject);
       private
      { Private declarations }
       public
      { Public declarations }
       end;

var
 MainForm: TMainForm;

implementation

{$R *.DFM}

{ Initialize controls }
procedure TMainForm.FormCreate(Sender: TObject);
var
 Ascii, IRow, ICol: Integer;
begin
{  Initialize FontCB ComboBox with font names   }
 FontCB.Items := Screen.Fonts;
{  Show current StringGrid font in FontCB's edit box   }
 FontCB.ItemIndex :=
     FontCB.Items.IndexOf(StringGrid1.Font.Name);
{  Insert characters into grid   }
 Ascii := 0;
   with StringGrid1 do
   for IRow := 0 to RowCount do
     for ICol := 0 to ColCount do
        begin
          Cells[ICol, IRow] := Chr(Ascii);
            Inc(Ascii);
```

```
    end;
{ Assign sample character and font }
 with StringGrid1 do
    CharLabel.Caption := Cells[Row, Col];
     CharLabel.Font.Name := StringGrid1.Font.Name;
end;

{ Change grid, edit box, and sample to selected font }
procedure TMainForm.FontCBChange(Sender: TObject);
begin
   StringGrid1.Font.Name := FontCB.Text;
   CopyEdit.Text := ''; { Optional: Erase current entries }
   CopyEdit.Font := StringGrid1.Font;
   CharLabel.Font.Name := StringGrid1.Font.Name;
end;

{ Close FontCB drop-down list on pressing Enter or Esc }
procedure TMainForm.FontCBKeyDown(Sender: TObject;
 var Key: Word; Shift: TShiftState);
begin
 if Key in [vk_Return, vk_Escape] then
   begin
    FontCB.DroppedDown := False;
      Key := 0;
     end;
end;

{ Grid double-click event handler }
procedure TMainForm.StringGrid1DblClick(Sender: TObject);
begin
 with CopyEdit, StringGrid1 do
    Text := Text + Cells[Col, Row];
end;
```

```
{   Select character on pressing Enter or Space   }
procedure TMainForm.StringGrid1KeyDown(Sender: TObject;
  var Key: Word; Shift: TShiftState);
begin
    if Key in [vk_Return, vk_Space] then
      StringGrid1DblClick(Sender);  // Same as double-click grid
end;

{   Show selected character   }
procedure TMainForm.StringGrid1SelectCell(Sender: TObject; Col,
  Row: Longint; var CanSelect: Boolean);
begin
  CharLabel.Caption := StringGrid1.Cells[Col, Row];
end;

{   File|Exit menu command   }
procedure TMainForm.FileExitClick(Sender: TObject);
begin
  Close;
end;

{   Help|About menu command   }
procedure TMainForm.HelpAboutClick(Sender: TObject);
begin
  AboutForm.ShowModal;
end;

{   Select button click handler   }
procedure TMainForm.SelectBitBtnClick(Sender: TObject);
begin
    StringGrid1DblClick(Sender);  // Same as double-clicking
   grid
end;

{   Copy selected characters to clipboard   }
```

```
procedure TMainForm.ClipBitBtnClick(Sender: TObject);
begin
  with CopyEdit do
    begin
     if SelLength = 0 then
        SelectAll;      { Select all text if none selected }
          CopyToClipboard;   { Copy selected text to
    clipboard }
      end;
end;

{   Clear text in copy-to edit box   }
procedure TMainForm.ClearBitBtnClick(Sender: TObject);
begin
  CopyEdit.Text := '';
end;

end.
```

يعـمـل الـ StringGrid component مـثـل الـ ListBox الثنائى الابعـاد . تحديد خاصية الـ RowCount عدد الصفوف ، وتحدد خاصية الـColCount عدد الاعمدة . على ابسط مستوى ، يمكنك استخدام هذه القيم للوصول الى بيانات الـ grid من خلال الـ Cells . على سبيل المثال ، يقوم برنامج الـ OnCreate للـ form بتعيين كل قيم الـ ASCII الممكنه الـ StringGrid1 object الخاص بالبرنامج بهذه العبارات (الـ Ascii ، IRow ، والـ ICol ومتغيرات Integer) :

```
Ascii := 0;
with StringGrid1 do
for IRow := 0 to RowCount do
  for ICol := 0 to ColCount do
    begin
     Cells[ICol, IRow] := Chr(Ascii);
       Inc(Ascii);
      end;
```

يمكنك ايضـاً التـوصـل الى grid strings من خـلال الـ Rows والـ Cols ، والتى تعـتبر single-dimensional string . كل عنصـر من الـ Rows والـ Cols يعتبر TStrings string-list object . لتحـميل بيانات صف من ملف قرص ، مثلاً ، يمكنك استخدام عبارة مثل :

```
with StringGrid1 do
Rows[0].LoadFromFile('C:\YourFile.Txt');
```

أو ، يمكنك تحميل صفوف متعددة بـ code مثل :

```
with StringGrid1 do
for I := 0 to RowCount do
 Rows[I].LoadFromFile('C:\YourFile.Txt');
```

عند عـمل إضـافات عديدة لصفوف واعـمدة الـ StringGrid (وبشكل عام عند إضـافة strings للـ TStrings والـ TStringList) ، ضع العـمليـات مـا بين بـ BeginUpdate و EndUpdate لإعـادة عـرض الـ object مـرة أخـرى . على سبيل المثال ، قد يكون المثال السابق ابسط عندما يصاغ كما يلى :

```
with StringGrid1 do
for I := 0 to RowCount do
with Rows[I] do
try
 BeginUpdate;
    Rows[I].LoadFromFile('C:\YourFile.Txt');
finally
 EndUpdate;
end;
```

من الصواب ان تستخدم الـ try-finally block كما هو موضح لتتأكد ان الـ EndUpdate قـد تم استـدعـاءه ، حتى اذا حـدثت أية exceptions فى العـبـارتين الواقعتين تحت try .

تعتبر أغلب إمكانيات الـ StringGrid هى نفسها إمكانيات الـ TStrings ، والتى اختبرتها بعمق ، لذا فلن أزيد هنا . ولكن ، يعرض البرنامج تقنية مفيدة ، وإن كـانت ليس لهـا عـلاقة ، لقـوائم الـ ComboBox التى قـد تجـدها مـعـاونة . فى الـ

OnKeyDown الخاص بـ FontCB object، اذا كان الـ Key Parameter الذى تم تمريره يعتبر vk_Return أو vk_Escape، يحدد الـ procedure خاصية الـ DroppedDown للـ ComboBox بـ False، ويضغط Enter أو Esc لإغلاق نافذة القائمة. وهذه اللمسة البسيطة تساعد على جعل واجهة تطبيق لوحة المفاتيح الخاصة بالـ CharGrid مألوفاً.

يوضح الـ OnKeyDown الخاص بالـ StringGrid كيفية برمجة مفاتيح الـ StringGrid object. وهنا، اذا كان الـ Key يساوى vk_Return أو vk_Space يستدعى الـ StringGrid1DblClick procedure، الذى يحاكى الضغط مرتين على الـ grid cell. حتى تعمل هذه التقنية، يجب أن تقوم بإضافة الـ OnDblClick الخاص بالـ grid.

فكرة: بشكل عام، وجدت أنه من الأسهل برمجة الـ event للفأرة أولاً (مثل الـ OnClick والـ OnDblClick)، ثم إضافة keyboard operations، إننى استدعى الـ mouse handlers من الـ OnKeyDown والـ OnKeyPress.

افكار للمستخدم الخبير

- تعود الـ Windows API ShellExecute على الفور- فهى لا توقف التطبيق الحالى لتشغيل آخر، كما يعتقد بالخطأ بعض واضعى البرامج. بعد أن يستدعى تطبيق A الـ ShellExecute لتشغيل تطبيق B، من الأمان أن تغلق تطبيق A بينما يظل B يعمل. إن الـ A والـ B تعتبر عمليات مشتركة مستقلة.
- تقوم الـ StringGrid عادة بتثبيت الصف والعمود الأولين التابعين لها فى مكان لأنك غالباً سوف تعرض headers فى هذه الخلية. اذا لم تكن فى حاجة الى headers (كما فى تطبيق الـ CharGrid)، حدد الـ FixedCols والـ FixedRows بصفر.
- يتم استدعاء الـ OnActivate event للـ form عندما تتلقى الـ form الـ input focus نتيجة تحول المستخدم مرة أخرى الى الـ form من واحدة أخرى فى

نفس التطبـيـق . إن الـ OnActivate event لا يعـتـبـر مـسـاويـاً لرسـالة الـ wm_Activate بالـ Windows .

● إن VisibleRowCount والـ VisibleColCount التـابعـان للـ StringGrid يشيـران الى كم الصفوف والأعمدة غير الثابتة فى الـ grid . تشير متغيرات الـ Row والـ Col الى العملية التى لها input focus .

● للكتـابة فى الـ cell فى الـ StringGrid ، حـدد الخـاصيـة الفـرعـيـة Options.goEditing بـ True . اذا أردت أن تمكن المستخدمين من ضغط الـ Tab للإنتقال من cell الى أخرى ، حدد الـ Options.goTab بـ True .

● لإعـادة تحديد حجم العمـود والصف بضـغط وسـحب الفـأرة ، حـدد الـ Options.goColSizing والـ Options.goRowSizing بـ True . لتـشـغـيـل ضـغط وسـحب الاعـمـدة والصـفـوف الى مـواضع جـديدة ، حـدد الـ Options.goRowMoving والـ Options.goColMoving بـ True .

● عندمـا يتم إنشـاء component وهو يقـوم بإدخـال نفـسـه فى قـائمـة الـ Components الخاصة للـ component التابع له . و الـ Ownership تعنى ان الـ child components يتم تدميرها عندما يتم تدمير ماليكها . ويحدد الـ parent component الـ display context. . بدون الـ parent component ، سـوف يصبح الـ control غير مرئياً . كل الـ components التى تنشئها فى وقت التصميم تابعه للـ form .

المشروعات التى يمكنك تجربتها

(١٠-١) : استـخـدام الـ ExecuteFile فى الـ FMXUtils unit التـابعـة لـ Delphi لإنشـاء control-panel application والذى يعـرض ايقونات التطبيق . يجب ان يكون المستخدمون قادرين على ضغط الايقونة لتشغيل البرامج . يمكنك توفير هذا البرنامج من خلال عدة utilities .

(١٠-٢) : اكـتب بديلاً للـ Windows Notepad ، يكـون أكـثـر مـلاءمـه لوضع الـ source code فى قائمة . يجب ان يستخدم برنامجك

monospace (أو يكون لدية أمر الـ Font لاختيار الـ fonts. فى اماكن اخرى من هذا الكتاب (الباب الثانى عشر، مثلاً)، سوف تعرف المزيد عن اوامر البحث والإبدال وكذلك الطبع، الذى يمكنك إضافتها الى text editor النهائى الخاص بك. ولكن، لا يزال هذا المشروع تمريناً مفيداً على استخدام الـ Memo.

(١٠-٣): اكتب code لنسخ النص المختار من والى Memos مختلفة.

(١٠-٤): اكتب برنامجاً يرتب الـ strings فى الـ Memo. (ملحوظة: يمكنك استخدام الـ ListBox الذى يقوم بترتيب البيانات بصورة تلقائية عندما تحدد خاصية الـ Sorted له بـ True، أو يمكنك كتابة code ليطبق الـ (Sort algorithm).

(١٠-٥): اكتب برنامجاً يقوم بالـ encrypts و decrypts لـ text files. أضف الحماية بكلمة المرور للبرنامج.

(١٠-٦): أضف ألوان للاحمر والاخضر والازرق الى الـ ColorDlg dialog.

ملخص:

- الـ StringGrid والـ Memo توفر multiple-line text-handling. ان الـ Memo يبدو وكأنك لديك الـ Windows Notepad utility فى صورة component. يعتبر الـ StringGrid من الناحية العملية (spreadsheet) ينظم قوائم الـ strings فى الاعمدة والصفوف.

- ان نسخةالـ Memo ذات الـ ١٦ بت يمكن ان تحمل الى ٣٢ K فقط من الـ Text. والنسخة الاحدث ذات الـ ٣٢ بت، التى تستخدم نوع بيانات الـ Object Pascal String، يمكن ان تحمل كم غير محدد من النص، ولكنه محدد من الـ Windows بـ ٦٤ K.

- يمكنك الوصول الى الـ Memo text على انه data object وحيد وذلك باستخدام خاصية الـ Text. للوصول الى الـMemo text على انه individual strings، استخدم خاصية الـ Lines. من الممكن ايضاً، رغم انه من غير المرغوب

فيه، ان تتوصل الى بيانات الـMemo كـ Char array وذلك باستدعاء methods مثل الـ GetTextBuf و SetTextBuf .

● الـ Text components ، مثل الـMemo والـEdit، توفر بصورة تلقائية القص والنسخ واللصق للـ clipboard . ولكن، يمكنك استدعاء الـ CopyToClipboard والـ CutToClipboard والـ PasteFromClipboard للـ Edit، والـMaskEdit، و الـMemo لأداء إنتقالات الـ clipboard تحت تحكم البرنامج.

● ان الـ WinExec function لازالت متاحة، ولكن غير موصى بها. لتشغيل تطبيقات من خارج برنامجك (أو لفتح ملفات فى تطبيقاتها الافتراضية)، استخدم الـ ShellExecute الخاصة بالـWindows API . ان اسهل الطرق لفعل هذا فى Delphi هى بإضافة الـ FMXUtils unit (الموجودة فى دليل الـ Demos\Doc\Filemanex التابع لـDelphi) باستخدام أمر الـ Project|Add to project . وكذلك اضف عبارة ;uses FMXUtils لقطاع الـ implementation. للـ form . يمكنك عندئذ استدعاء الـExecuteFile المتوفرة فى الـ FMXUtils . يقوم الـExecuteFile باستدعاء الـ ShellExecute الخاص بالـWindows API .

● استخدم الـScrollBox على انه range-selection object. استخدم الـScrollBox لتوفير سطح قابل للـ scrollable يشبه الـPanel يمكن ان يحمل controls اخرى.

● فى container object. مثل الـ form أو الـ ScrollBox، تحتوى قائمة Components array بكل الـ components التابع لهذا الـ object. يحتوى الـ Controls array على كل الـ component التى تعتبر child windows . هذه الـ arrays لا تحتوى نفس الـ component بالضرورة. استخدم البرمجة الموجودة بتطبيق الـContComp الخاص بهذا الباب لفحص علاقات الـ controls والـcomponents بين الـ objects. ان فهم هذه العلاقات يمكن ان يساعدك على كتابة code فعالة- على سبيل المثال، loop توقف عمل كل child controls فى الـ ScrollBox دون التأثير على objects أخرى بالـ form .

• تقدم الـ StringGrid عمود وصف لتخزين starins وبيانات object أخرى. استخدم الـ Cells array الثنائى الأبعاد للتوصل الى Cell منفردة. استخدم الـ Rows arrays والـ Cols الأحادية البعد للوصول الى بيانات الـ grid على أنها قوائم strings.

• إن التعامل مع ملفات القرص، اسماء الملفات، والدلائل، والأمور المتعلقة بها يعتبر جانباً هاماً فى أغلب مشاريع تطوير البرمجيات. فى الباب القادم، سوف تعرف المزيد عن directory navigation الـ file components.



الباب الحادى عشر
Navigating Directories and Files

محتويات هذا الباب:

- Components
- إنشاء الـ directory dialog
- تطوير الـ directory-based utilities
- ملفات الـ Drag-and-drop
- قراءة وكتابة ملفات الـ ini.

يعتبر الـ directory و file management امراً هاماً فى التطبيقات المؤسسة على ملفات . حتى ابسط utility تحتاج الى قراءة وكتابة مواصفات اختيارية فى ملف البدء (ini.) الخاص بالـ Windows ، وفى الغالب تحتاج البرامج الى تقنية واحدة على الاقل فى التعامل مع الملف والموضحة فى هذا الباب .

يقدم Delphi اربعة components للـ directory والملف والتى يمكنك استخدامها لإنشاء directory-navigation dialogs و File Selection . فى هذا الباب ، تستخدم هذه الـ components لإنشاء File Selection dialog وبرنامجاً يمكن ان يقوم بتشغيل أى برنامج تنفيذى . سوف تعرف ايضاً كيفية استخدام الـ TIniFile object لقراءة وكتابة ملفات الـ ini. . وعلى سبيل المثال التعامل مع ملفات الـ ini ، يوضح هذا الباب الاعمال الداخلية لبرنامج utility معقد نسبياً ، وهو الـ SysColor ، وهو يمكنه ان يغير الالوان الموجودة على الشاشة مثل

ظلال الازرار و title bars . ان البرنامج يقرأ ويكتب عناصر اللون فى ملف الـ ini. الخاص بالـ Windows .

Components :

فيما يلى directory File components و Delphi

• **DirectoryListBox**: يعرض ويسمح للمستخدمين بتغيير الدليل الحالى والأدلة الفرعية. فى اغلب الحالات، إنك تربط الـ DirectoryListBox بالـ FileListBox لعرض ملفات عندما يتصفح المستخدمين دليل لقرص. الـ Palette : Win3.1 .

• **DirectoryListBox**: يعرض ويسمح باختيار كل drives المتصلة بالنظام. إنك تربط بشكل طبيعى الـ DirectoryListBox بالـ DirectoryListBox متى يتمكن المستخدمون من الإنتقال الى drives آخر وتصفح ادلته . Palette : Win3.1 .

• **FileListBox**: يعرض ملفات الدليل الحالى. يمكن ان يعرض الـ FileListBox كل الملفات أو تلك التى تتماشى مع واحداً أو أكثر من الـ wildcard filters، مثل الـ pas.* والـ txt.* . إنك تربط الـ FileListBox بالـ Edit لتزود المستخدمين بوسائل إدخال وتحرير اسماء الملف. Palette : Win3.1 .

• **FilterComboBox**: قائمة الـ wildcard filters التى يمكن للمستخدمين الأختيار منها لتحديد انواع الملفات التى يعرضها الـ FileListBox المرتبط به. Palette : Win3.1 .

Note

ملحوظة: بالرغم من ان الـ components السابقة توجد فى دليل الـ Win3.1 ، فهى مازالت ذات قيمة فى إنشاء مديرى ملفات مخصصة و controls أخرى لتصفح دليل آخر. تعمل الـ components بشكل مثالى مع الـ Windows 95, 98,NT .

إنشاء Directory Dialog :

تقدم components الملف والدليل الاربعة الخاصة بـ Delphi مجموعة ادوات Erector التى يمكنك استخدامها لإنشاء File Selection dialogs . فى

هذا الفصل، ستقابل كل الـcomponents ، ثم تستخدمهم لإنشاء File dialogs Selection الـــذى يمكنـــــك وضعـــه للعمـــــل فى اوامر الـ File|Open ، والـ File|Save ، والاوامر الأخرى للتعامل مع الملف فى تطبيقك.

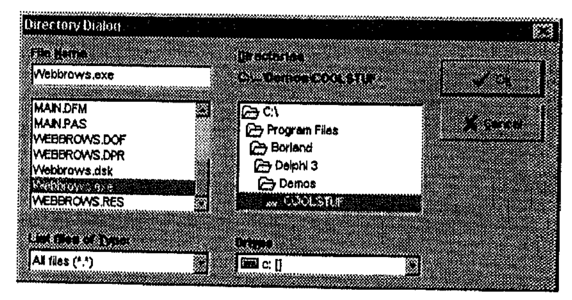
إنك تدرك غـالبـاً ان الـ Delphi يقـدم الـ OpenDialog والـ SaveDialog على الـ Dialogs palette (انظر الباب الثانى عشر). استـخـدم التقنيات التـالية عندمـا تحتـاج ان تعتـاد على ذلك- على سبيل المثـال، لإضافة wildcard filter editor، كواحد من المشروعات المقترحة فى هذا الباب.

DirectoryListBox :

يوضح شكـل (١١-١) الـ dialog box الذى تنشئــه مـؤخـراً . فى هذا الفـــصل. يعـــرض الـ DirectoryListBox تخطيط لشـــجــرة دليل . يمكن للمستخدمين الضغط مرتين واستخدام لوحة المفاتيح لاختيار ادلة فى هذه النافذة.

يمكنك استـخـدام الـ DirectoryListBox بمفـرده فى نافـذة، ولكنك عـادة تريد عـرض ملفـــات فى الأدلة المخـتــارة. لتـفـعل هذا، قم بتـعـيين اسم الـ FileListBox لخـاصـيـة الـ FileList التـابعـة للـ DirectoryListBox . هذه هى اسهل طريقة ويمكـن اداؤها فى نافـذة الـ Object Inspector، ولكن يمكنك القيام بالتـعـيين فى وقـت التـشـغـيـل كـالآتى. أضـف الـ DirectoryListBox والـ FileListBox على الـ form، ثم إدخل هذه العبارة فى الـ OnCreate الخاص بالـ form:

```
DirectoryListBox1.FileList := FileListBox1;
```



شكل (١١-١): فى هذا الباب dialog box يستخدم components الملف والدليل الاربعة لـ Delphi: DirectoryListBox، DriveComboBox ، FileListBox ، وFilterComboBox

يوضح Delphi قوائم الملف والدليل الحالية فى وقت التصميم، ولكن يجب عليك تشغيل البرنامج لاختيار ملفات وادلة جديدة، ولتحريك النوافذ.

لإظهار المسار الذى تم اختياره حالياً كـ string، أضف Label object على الـ form (عادة، يجب ان يكون فوق الـ DirectoryListBox). قم بتعيين اسم الـ Label لخاصية الـ DirLabel التابعة للـ DirectoryListBox. أو، لتقوم بالربط اثناء وقت التشغيل، إدخل هذه العبارة فى الـ OnCreate الخاص بالـ form:

```
DirectoryListBox1.DirLabel := Label1;
```

يقوم الـ DirectoryListBox بتعيين مسارات الدليل، بما فى ذلك الـ drive الحالى، ليضاف الى الـ Caption الخاص بالـ Label. لعرض مسارات متداخلة بع يستبدل الـ component الادلة الجذرية الاساسية بزر بيضاوى. هذا يجعل الـ Label قصير نسبياً (حوالى ٢٤ حرف كحد أقصى، أو عرض الـ DirectoryListBox تقريباً). على سبيل المثال، يعرض الـ DirectoryListBox هذا المسار:

```
C:\Program Files\Borland\Delphi 4\Demos\COOLSTUFF
```

على انه الـ Caption المختصر:

```
C:\...\Demos\COOLSTUFF
```

خاصية الـ Drive تعطى حرف الـdrive المختار حالياً. على سبيل المثال، يحدد حرف الـ C على انه حرف الـ drive:

```
C := DirectoryListBox1.Drive;
```

استخدم خاصية الـ Directory لتحديد المسار الذى يتم اختياره حالياً.

لعرض اسماء مسارات كاملة غير مختصرة بزر بيضاوى، لا تحدد اسم Label لخاصية الـ DirLabel. بدلاً من ذلك، أضف Label على الـ form وفى OnChange الخاص بالـ DirectoryListBox، قم بتعيين الـ Directory string الخاص بالـ Caption الخاص بالـ Label.

DriveComboBox :

اذا لم تكن تريد السماح بالوصول الى drive معين فقط ، فإنك غالباً تريد إضافة الـ DriveComboBox وربطه بالـ DirectoryListBox . أضف DriveComboBox على الـ form (غالباً ما يكون أسفل الـ form) . عين إسم DriveComboBox لخاصية الـ DirList التابعة للـ DriveComboBox ، أو استخدم هذه العبارة فى OnCreate الخاص بالـ form :

```
DriveComboBox1.DirList := DirectoryListBox1;
```

عندما يختار المستخدمون drive مختلف ، يقوم الـ DriveComboBox تلقائياً بتحديث شجرته . اذا ربطت ايضاً الـ FileListBox بالـ DirectoryListBox ، يتم تحديث قائمة الملف ايضاً .

ان القائمة الكاملة للـ drives المتاحة متوفرة من خلال خاصية الـ Items للـ DriveComboBox وهو object من نوع الـ TStrings . يمكنك تعيين هذه القائمة لأى خاصية TStrings أخرى ، أو لمتغير TStringList . على سبيل المثال ، لعرض كل drive متاحة ، أضف الـ ListBox على الـ form ، واضف هذه العبارة لـ OnCreate الخاص بالـ form :

```
ListBox1.Items := DriveComboBox1.Items;
```

FileListBox :

كما يمكن ان تتوقع ، يعرض الـ FileListBox اسماء ملفات فى الدليل الحالى . فى اغلب الاوقات يكون الـ DirectoryListBox مرتبط بالـ FileListBox . بهذه الطريقة ، تتغير القائمة تلقائياً عندما يتصفح المستخدمون الأدلة .

إنك غالباً ما تضيف ايضاً إثنين من الـ components : الـ Edit والـ FileListBox . أضف الـ Edit الآن (يوضح الفصل التالى كيفية استخدام الـ FileListBox) . عادة ما يكون الـ control فوق الـ FileListBox . قم بتعين Name للـ Edit لخاصة الـ FileEdit التابعة للـ FileListBox ، أو إدخل هذه العبارة فى الـ OnCreate الخاص بالـ form :

```
FileListBox1.FileEdit := Edit1;
```

عندما تقوم بتشغيل البرنامج، يعرض الـ Edit أولاً الـ drive *.* ، الذى يختار كل الملفات. عندما يختار المستخدمون اسماء الملفات، يقوم الـ FileListBox بإدخالها فى نافذة الـ Edit.

لإنشاء Read only Edit لا يستطيع المستخدمون تعديله، ولكن يستمر فى عرض اختيار الملف الحالى، حدد خاصية الـ ReadOnly لـ Edit بـ True. لمنع المستخدمين ايضاً من الانتقال للـ control وإبراز نصه، حدد الـ Enabled بـ False.

إجعل الملف المختار فى خاصية الـ FileName للـ FileListBox. يتضمن هذا الـ string مسار وحرف الـ drive الحالى. على سبيل المثال، لاختيار الملفات بالضغط مرتين، قم بإنشاء الـ OnDblClick للـ FileListBox ، وإدخل هذه العبارة:

```
ShowMessage(FileListBox1.FileName);
```

قم بتشغيل البرنامج واضغط مرتين اسماء الملفات لعرضها فى نافذة ShowMessage. لاختيار ملفات ايضاً بضغط Enter، قم بإنشاء الـ OnKeyPress الخاص بالـ FileListBox، وإدخل هذا الـ code لإستدعاء double Click procedure اذا كانت الـ Key للـ carriage-return مساوية للـ control code :

```
if Key = #13 then
  FileListBox1DblClick(Sender);
```

اذا كنت لا تحتاج إسم المسار كاملاً فى الـ FileName string الخاص بالـ FileListBox ، قم بتمريره الى احدى الـ File-Management functions . بـ Delphi: ExtractFileExt، ExtractFileName ، و ExtractFilePath .

لتجربة هذه الـ functions أضف ثلاثة Labels على الـ form (يقوم Delphi بتسميتهم بـ Label2، Label3، و Label4). قم بتعيين اجزاء من مسارات الملف المختار بهذا الـ code فى الـ OnDblClick الخاص بالـ FileListBox :

```
with FileListBox1 do
begin
```

```
    Label2.Caption := ExtractFileExt(FileName);
     Label3.Caption := ExtractFileName(FileName);
     Label4.Caption := ExtractFilePath(FileName);
  end;
```

يستطيع المستخدمون اختيار ملفات متعددة فى FileListBox اذا كانت خاصية الـ MultiSelect له محددة بـ True. بهذا التحديد، يمكن للمستخدمين ابراز ملفات متعددة بضغط مفاتيح الـ Shift والـ Ctrl أثناء ضغط الفأرة. اذا اخترت هذا الخيار، فإنك تحتاج ان تكتب code وتدخل زر OK لتوفير قائمة الاختيار للتطبيق.

يقوم الـ Edit المرتبط بالـ FileListBox فى خاصية الـ FileEdit بعرض احدث اسم ملف تم اختياره عندما تحدد الـ MultiSelect بـ True. ولكن بدلاً من الـ Edit، يمكنك إضافة الـ ListBox فى الـ form لعرض اسماء الملفات المختارة.

توضح القائمة (١١-١) كيفية إنشاء قائمة اختيار اسم ملف متعدد. قم بإضافة ListBox آخر فى الـ form (يقوم Delphi باعطاءه اسم ListBox2)، وإدخل BitBtn بخاصية الـ BitBtn له محددة بـ bkOk. إدخل البرمجة الموجودة فى القائمة (١١-١) فى OnClick للـ BitBtn.

القائمة (١١-١) : الـ OnClick هذا يوضح كيفية
إنشاء قائمة اختيارات متعددة فى الـ FileListBox

```
  procedure TForm1.BitBtn1Click(Sender: TObject);
  var
     I: Integer;
  begin
   ListBox2.Clear;
     if FileListBox1.SelCount > 0 then
     for I := 0 to FileListBox1.Items.Count - 1 do
       if FileListBox1.Selected[I] then
           ListBox2.Items.Add(FileListBox1.Items[I]);
    end;
```

اذا كان الـ SelCount أكبر من (0)، يكون المستخدم قد اختار اسم ملف واحد على الأقل. تستخدم الـ for loop خاصية الـ Count فى الـ Items لتحدد إجمالى أســـم الملفـــات الذى يقوم الـFileListBox بعرضه. لكل عنصر N، اذا كانت الـ Selected[N] بـ True، يضيف البرنامج هذا العنصر لقـــائمة الـ Items الخاص بالـListBox2. إستبدل الـ ListBox فى القائمة (١١-١) بـ TStringList object اذا كنت تحتاج قائمة اختيار إسم ملف ليست مرتبطة مثل الـ ListBox .

يمكنك تحسين فعاليةالـ code الموجودة فى القائمة (١١-١) باستخدام عبارة with، ولكننى وضعت قبل الخصائص اسم الـ object حتى اوضح ما تفعله العبارات. لفعل هذا التغيير، استبدل العبارات الواقعة بعد استدعاء الـListBox2 لـ Clear method بما يلى:

```
with FileListBox1, Items do
if SelCount > 0 then
for I := 0 to Count - 1 do
 if Selected[I] then
     ListBox2.Items.Add(Items[I]);
```

: FilterComboBox

استخدم الـ FilterComboBox لتوفير قائمة من الـ wildcard filters التى يمكن الاختيار منها مثل الـ pas.* والـ ini.* . قم بإضافة الـ FilterComboBox فى على الـ form (يأتى عادة تحت الـ FileListBox). لتحديث قائمة الملف الحالى عندما يختار المستخدمون الـ drive، عين اسم الـ FileListBox لخاصية الـ FileList للـFilterComboBox. أو إدخل هذه العبـــارة فى الـOnCreate الخاص بالـ form:

```
FilterComboBox1.FileList := FileListBox1;
```

افتح الـ Filter editor الخاص بـ Delphi بضغط الزر البيضاوى الواقع بعد خاصية الـ Filter للـ FilterComboBox. أضف الـ filters فى الـ editor بهذه الصورة:

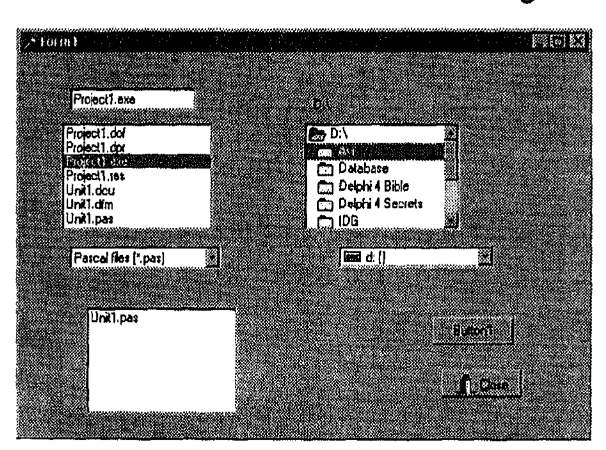
```
All files (*.*)         | *.*
Text files (*.Txt)      | *.Txt
Pascal files (*.pas )   | *.pas
```

تظهر strings العمود الايسر في الـ FilterComboBox . تعتبر strings العمود الايمن هي الـ filters الفعلية وقد تم تمريرها للـ FileListBox . اذا كان الـ Edit object متعلق به ، فإن اختيار filter جديد يؤدى ايضاً الى إدخاله في نافذة الـ Edit .

وضع dialogs الأدلة معاً:

على القرص المدمج: اذا كنت مستمر فى المتابعة ، فإن الـ form تبدو مثل الـ dialog المختلط فى شكل (١١-٢) . فى هذا الفصل ، تقوم بترتيب هذه الفوضى بـ dialog اختيار ملف يبدو فى مظهر أفضل والذى يمكنك استخدامه فى كثير من العمليات من التعامل مع الدليل والملف .

توضح القائمة (١١-٢) الـ source code للـ DirDlg (يمكنك العثور عليها على القرص المدمج فى دليل Source\DirEx) . لا يوجد كثير من الـ code فى هذه الـmodule - تعتبر اغلب عمليات الـ dialog تتم بصورة بلقائية وتنتج عن ربط objects الدليل والملف .



شكل (١١-٢): الأولى (والفوضوى) الذى تم إنشاءه فى dialog box الفصول السابقة موضحاً components دليل وملف Delphi

القائمة (١١-٢) الـ DirDlg module فى الدليل الفرعى DirEx

```
unit Dirdlg;
```

```
interface

uses
 SysUtils, Windows, Messages, Classes, Graphics,
    Controls, Forms, Dialogs, StdCtrls, Buttons, FileCtrl;

type
 TDirDlgForm = class(TForm)
      FileListBox: TFileListBox;
         DirectoryListBox: TDirectoryListBox;
         DriveComboBox: TDriveComboBox;
         FilterComboBox: TFilterComboBox;
         FileNameEdit: TEdit;
         FileNameLabel: TLabel;
         DirectoriesLabel: TLabel;
         DirLabel: TLabel;
         ListFilesLabel: TLabel;
         DrivesLabel: TLabel;
         OkBitBtn: TBitBtn;
         CancelBitBtn: TBitBtn;
         procedure FileListBoxDblClick(Sender: TObject);
     private
      { Private declarations }
        public
      { Public declarations }
        end;

var
 DirDlgForm: TDirDlgForm;

implementation

{$R *.DFM}
```

```
procedure TDirDlgForm.FileListBoxDblClick(Sender: TObject);
begin
  OkBitBtn.Click;
end;

procedure TDirDlgForm.DriveComboBoxChange(Sender: TObject);
begin
  DirectoryListBox.Drive := DriveComboBox.Drive;
     FileListBox.Drive := DriveComboBox.Drive;
     FileListBox.Directory := DirectoryListBox.Directory;
end;

end.
```

يوضح برنامج اختيارى فى دليل الـ DirEx كيفية استخدام الـ DirDlg dialog. تقوم الـ ShowModal function الخاصة بالـform بإدخال قيمة الـ ModalResult، والتى يمكن للتطبيق ان يفحصها ليتعرف الـ dialog على سبيل المثال، تعرض القائمة (١١-٣)، المأخوذة من البرنامج الاختيارى بالـDirEx، إسم ملف مختار فى الـdialog ShowMessage اذا كانت الـShowModal تدخل mrOk. لكى يحدث هذا، تقوم ازرار الـ OK و الـ Cancel الخاصة بالـ dialog بتحديد خصائص الـModalResult لها بـ mrOk و mrCancel على التوال. يمكنك ايضاً تعيين قيم للـ ModalResult فى وقت التشغيل بعبارة مثل:

```
ModalResult := mrOk;
```

القائمة (١١-٣): الـShowMessage يعرض file - Selection dialog [راجع شكل (١١-١)]

```
procedure TMainForm.TestBitBtnClick(Sender: TObject);
begin
  with DirDlgForm do
    if ShowModal = mrOk then
      ShowMessage('Selected file = ' + FileNameEdit.Text);
end;
```

تطوير الـ Directory - Based Utilities:

عندمـا تقـوم بإنشـاء forms مـثل file - Selection dialog فى الفـصل السابق، فإنك قد تحتاج ان تستخدمها فى تطبيقات اخرى. لتنظيم الـ source code لتطبيقك، من الافضل دائماً ان تنشئ دليلاً منفصلاً لكل module لكل برنامج. يمكن ان يستخدم برنامجاً واحداً أو أكثر هذه الـ modules بإضافتها الى المشروع.

كـمـثـال على هذا المفـهـوم، ولتـوضـيح كـيـفـيـة اسـتـخـدام- dialog file Selection فى الـ utility، يقوم الـ DirExec، بمضـاعفة إمكانيات الـ Explorer ليـقـوم بتـشـغيـل أى ملف الـexecutable code. يسـتطيع البـرنامج ايضاً فـتح أى ملف فى قاعدة بيانات سجل الـ Windows. على سبيل المثال، يفتح الـ DirExec الـ Notepad لعرض text file والذى ينتهى بالإمتداد .txt. .

على القرص المدمج: حاول إنشاء هذا التطبيق بنفسك- وهذا يمنحك الخبـرة فى اسـتـخدام الـ Project Manager الخاص بـDelphi لإقامة مشروعات متعددة لتطوير الـ module. قم بإنشاء دليل لبرنامجك، وابدأ مشروعاً جديداً. انظر الـProject Manager، واضغط زر الـ Add. إنتقل إلى نسـخـة من دليل الـ Source\DirEx من على القـرص المدمج، واختـر ملف DirDlg.pas file. هذا يضيف الـ DirDlg module لمشروع البـرنامج. يجب ايضـاً ان تضيـف اسم module unit لأمـر الـ uses الخـاص بالبـرنامج المضـيف. يمكنك عندئذ إضافة ازرار أو أوامر المشروع لفتح الـ dialog لاختيار اسماء ملفات. استـخـدم القـائمة (١١-٤) كمـرشـد لك. بعـد القـائمة، سـوف اوضح كـيف يقـوم البرنامج بتشغيل التطبيقات وفتح الملفات المسجلة. يوضح الشكل (١١-٣) عرض البرنامج.

القائمة (١١-٤): DirExec\Main.pas

```
unit Main;

interface

uses
  SysUtils, Windows, Messages, Classes, Graphics,
```



```
    Controls, Forms, Dialogs, StdCtrls, Buttons, ShellAPI,
     DirDlg;

type
    TMainForm = class(TForm)
      RunBitBtn: TBitBtn;
        BitBtn2: TBitBtn;
        procedure RunBitBtnClick(Sender: TObject);
      private
     { Private declarations }
      public
     { Public declarations }
      end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}

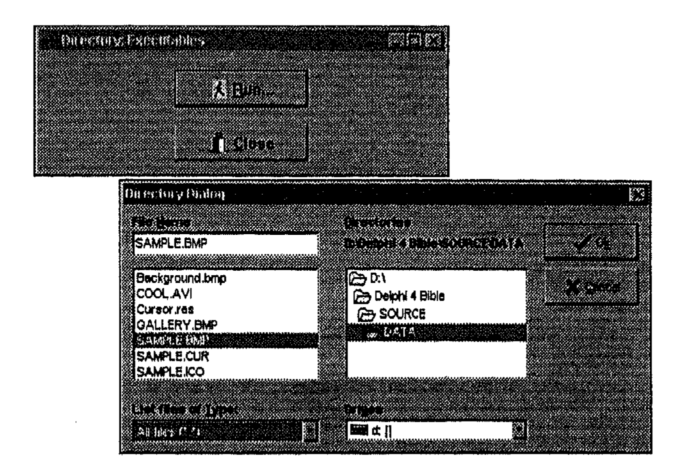
function ExecuteFile(const FileName, Params, Dir: String;
  ShowCmd: Integer): THandle;
begin
    Result := ShellExecute(Application.MainForm.Handle,
   nil,
      PChar(FileName), PChar(Params), PChar(Dir),
       ShowCmd);
end;

procedure TMainForm.RunBitBtnClick(Sender: TObject);
begin
  with DirDlgForm do
    if ShowModal = mrOk then
```



```
    if ExecuteFile(FileNameEdit.Text, '',
      DirectoryListBox.Directory,SW_SHOW) <= 32 then
        MessageDlg('Unable to open file or program',
          mtError, [mbOk], 0)
          else
      Application.Minimize;
  end;

  end.
```



شكل (١١-٣): يوضح تطبيق الـ DirExec كيفية استخدام dialog اختيار الملف الخاص بهذا الباب لتشغيل برامج اخرى وفتح ملفات مسجلة

فى الباب السابق، عرفت كيفية استخدام الـ ExecuteFile function الخاصة بالـ FMXUtils unit لتشغيل برامج اخرى. (توجد الـ unit فى دليل الـ Demos\Doc\Filemanex الخاص بـ Delphi). تقوم هذه الـ function باستدعاء Windows API function، وهى الـ ShellExecute، وهى متاحة دون استخدام الـ FMXUtils function. يمكن للـ ShellExecute ايضاً ان تفتح أى ملف مسجل- على سبيل المثال، تلك الملفات التى لها اسماء تنتهى بـ .txt و .bmp..

مثال الـ DirExec [راجع القائمة (١١-٤)] يتضمن function، وهى الـ ExecuteFile، والتى يمكنك قصها ولصقها فى أى تطبيق. تقوم الـ function باستدعاء الـ ShellExecute بخمسة parameters:

* الـhandle التابع للتطبيق الحالى (موجود فى خاصية الـHandle فى الـ MainForm الخاص بالـApplication).

* للعملية الاختيارية (حدد الـ 'Print' هنا لطباعة ملف بدلاً من فتحه).

* ثلاث null-terminated strings.

لان الـ ShellExecute يعتبر Windows API function، فلا يمكنك تغذيته بـ Pascal strings. تمثل الـ string الثلاثة: اسم الملف، أى parameters، ومسار الدليل.

يقوم الـ ShellExecute بإدخال الـhandle الحالى للتطبيق المفتوح. اذا كان صالحاً، فإن هذه القيمة تكون اكبر من ٣٢، وهذه الحقيقة يستخدمها الـ OnClick لزر الـRun الخاص بالبرنامج الاختيارى لعرض رسالة خطأ. لاحظ الاستدعاء للـ Application.Minimize قرب نهاية القائمة. هذه الخطوة اختيارية، ولكن تزيح نافذة البرنامج بعيداً عن طريق التطبيق المختار.

ملفات Drag-and-Drop:

أى برنامج يستطيع عرض ملف نص يجب أن يتعامل مع خدمات الـ Drag-and-Drop حتى يتمكن المستخدمون من سحب اسماء الملفات من الـ Windows File Manager وإسقاطها فى الـ Memo أو أى component آخر بالتطبيق. ان Delphi لا يقدم خدمات الـ drag-and-drop مباشرة، ولكن يمكنك إضافة هذه الميزة ببذل القليل من الجهد.

الخطوة الأولى هى ان تخبر الـ Windows ان نافذة برنامجك يمكن ان تقبل ملفات drag-and-drop افعل هذا فى الـOnCreate الخاص بالـ form بإضافة هذه العبارة:

```
DragAcceptFiles(Handle, True);
```

وايضاً، أضف الـ ShellAPI unit فى أمر الـ uses للـ module. يقوم الـ ShellAPI بتعريف الـ DragAcceptFiles والـ parameter الخاصة بها:

```
procedure DragAcceptFiles(Wnd: HWnd; Accept: Bool);
```

• **Wnd**: handle للنافذة الحالية، وهو الذى يحدد النافذة التى تتقبل عملية الإسقاط والسحب للملفات غالباً، يجب ان تمرر خاصية الـHandle الخاصة بالـ form التى تتقبل إسقاط الملف.

• **Accept**: حدد هذا الـ parameter بـ True لإخبار الـ Windows ان هذه النافذة تقبل اسماء الملف التى يتم سحبها واسقاطها. حدد هذا الـ parameter بـFalse لفصل عمليات السحب والاسقاط. يوصى Microsoft بالقيام بهذا قبل إنتهاء البرنامج - فى تطبيقات Delphi، يمكنك إيقاف السحب والاسقاط فى الـOnCreate الخاص بالـ form.

فى أية نافذة تستدعى الـ DragAcceptFiles، عندما تقوم بسحب اسم ملف من الـWindows File Manager الى ذلك النافذة وتطلق زر الفأرة، يرسل الـ Windows الى النافذة رسالة wm_DropFiles. ومع الرسالة يأتى اسم الملف لتلقى هذه الرسالة، ادخل الـ event handler فى القطاع الـ private للـ form class الخاصة بك. (يمكن ان يذهب التعريف الى أى مكان بالـ class، ولكنه غالباً ما يكون فى القطاع الـ private أو الـ protected. على سبيل المثال، اضف السطور التالية بين القطاعين الـ private والـ public فى تعريف الـ form class:

```
protected
  procedure WMDropFiles(var Msg: TMessage);
      message wm_DropFiles;
```

تكون الـ Protected procedures من نوع الـ never-never land بين القطاعين الـ private and public. يمكن للـclass method و الـ method فى الـ classes المشتقة فقط ان تصل إلى الـ protected methods. وهذا يختلف عن التعريف الـ private، والتى يمكن ان يصل إليها اعضاء الـ class فقط، ولكن ليس لأى classes مشتقة. تستطيع ايه عبارة الوصول لقطاع الـ Public للـ class كالغسيل المنشور بين البيوت. العناصر الـ protected فهى فى الردهة- يجب ان يكون داخل البيت حتى تحصل عليها. اما التعريف الـ private فهى فى حجرتك الخاصة. فطالما ابقيت الباب مغلقاً بقفل، فلا يستطيع احد ان يمسهم الا انت.

قم بتنفيـذ الـ WMDropFiles method لأداء اعمـال عندمـا يقـوم المستخدمون باسقاط اسم ملف على النافذة . يجب ان يكون للـ procedure الـformat العامة الموضحة فى القائمة (١١-٥). يشير التعليق الى المكان الذى يمكنك ان تؤدى فيه اعمالاً مع اسماء الملفات التى يتم اسقاطها .

القائمة (١١-٥): نفذ الـ WMDropFiles method بهذا النوع من الـ code

```
{  Handle wm_DropFiles message  }
procedure TMainForm.WMDropFiles(var Msg: TMessage);
var
 Filename: array[0 .. 256] of Char;
begin
 DragQueryFile(
     THandle(Msg.WParam),
        0,
        Filename,
        SizeOf(Filename));
      {  ... Perform action with dropped filename  }
    DragFinish(THandle(Msg.WParam));
end;
```

فى ملفـات الـ WMDropFiles ، يحمل null-terminated string ، وهو الـ FileName ، اسم الملف المسقط . يمكنك ان تجعل هذا الـ string اطول اذا لزم الأمر ، ولكن الـ ٢٥٦ رمزاً يجب ان يكونوا كافيين حتى لاكثر اسماء المسارات المتداخلة عمقاً . استدع الـ DragQueryFile للحصول على المعلومة المسقطة . يعرف الـ ShellAPI هذه الـ function بـ:

```
function DragQueryFile(Drop: THandle; FileIndex: Word;
 FileName: PChar; cb: Word): Word;
```

- **Drop :** handle للـ structure الداخلى الذى يحتوى معلومات عن اسماء الملفات التى تم سحبها وإسقاطها . غالباً يجب ان تمرر الى الـ Drop حقل الـ Msg.WParam من رسالة الـ WMDropFiles الملقاه فى الـ THandle . object.

- **FileIndex**: حــدد هذا الـ parameter بـ $FFFF (يســاوى 1- بالكســر العشرى) لطلب عدد اسماء الملفات المسقطة. قم بتمرير قيمة من صفر الى عدد اســمــاء الملفــات المســقطة ناقص واحــد لنسخ الاســمــاء فى الـ FileName parameter.

- **Filename**: Char array كبير بما يكفى لان يحمل اسم مسار كامل، عــادة ٢٥٦ بايت على الأقل. إدخل null-terminated string فى الـ array، أو قم بتمرير nil لطلب عدد الملفات المسقطة. قم بتمرير nil ايضاً لطلب حجم ملف بعينه بالـ bytes.

- **cb**: حجم الـ Filename Buffer بالـ bytes.

اذا قــمــت بتــمـــريــر (1-) لـ FileIndex parameter، تقـــوم الـ DragQueryFile بإدخال عدد الملفات المسقطة. قم بتعريف متغير Word لحمل نتيجة الـ function:

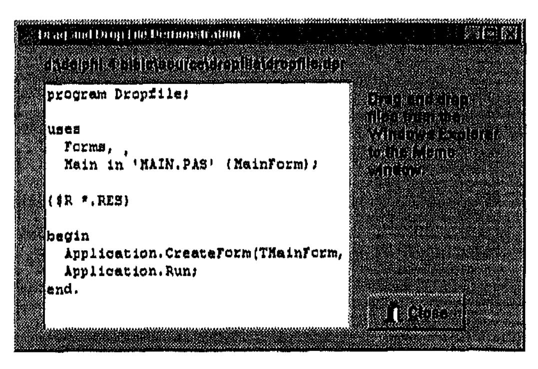
```
var
  NumFiles: Word;
```

بعد ذلك، استدع الـ DragQueryFile لتحديد عدد اسماء الملفات التى تم إسقاطها فى النافذة:

```
NumFiles := DragQueryFile(
  THandle(Msg.WParam),
    $FFFF,
    nil,
    0);
```

On the CD-ROM **على القرص المدمج**: فى كثير من الحالات، انك تحتاج ان تقبل اسم ملف واحد فقط. كمثال عملى على هذا، يوضح تطبيق الـ DropFile على القرص المدمج (فى دليل الـ Source\DropFile) كيفية برمجة Memo ليقبل اسماء ملفات السحب والاسقاط. قم بتشغيل الـ DropFile واسحب أى اسم ملف نص من الـ Windows File Manager الى نافــذة الـ Memo بالبرنامج، والتى تقرأ الملف من القرص وتعرض سطوره.

يوضح شكل (١١-٤) عرض البرنامج. توضح القائمة (١١-٦). الـ source code.



شكل (١١-٤): برنامج الـ DropFile يوضح ملف مشروع الـDropFile.dpr، والذى قمت بسحبه لنافذة الـ Memo من الـ Windows File Manager

القائمة(١١-٦): DropFile\Main.pas

```
unit Main;

interface

uses
  SysUtils, Windows, Messages, Classes, Graphics,
    Controls, Forms, Dialogs, StdCtrls, Buttons, ShellAPI;

type
  TMainForm = class(TForm)
     Memo1: TMemo;
       BitBtn1: TBitBtn;
       Label1: TLabel;
       FileNameLabel: TLabel;
       procedure FormCreate(Sender: TObject);
       procedure FormDestroy(Sender: TObject);
     private
    { Private declarations }
     protected
```

```
        procedure WMDropFiles(var Msg: TMessage);
           message wm_DropFiles;
           public
        { Public declarations }
         end;

var
 MainForm: TMainForm;

implementation

{$R *.DFM}

{ Handle wm_DropFiles message }
procedure TMainForm.WMDropFiles(var Msg: TMessage);
var
 Filename: array[0 .. 256] of Char;
begin
 DragQueryFile(
      THandle(Msg.WParam),
         0,
         Filename,
         SizeOf(Filename));
        with FileNameLabel do
     begin
Caption := LowerCase(FileName);
Memo1.Lines.LoadfromFile(Caption);
 end;
    DragFinish(THandle(Msg.WParam));
end;

{ Tell Windows this window can accept drag-and-drop files }
procedure TMainForm.FormCreate(Sender: TObject);
```

```
begin
  DragAcceptFiles(Handle, True);
end;

{ Disable drag-and-drop files (recommended) }
procedure TMainForm.FormDestroy(Sender: TObject);
begin
  DragAcceptFiles(Handle, False);
end;

end.
```

لتشغيل سحب واسقاط اسماء الملف، يقوم OnCreate الخاص الـ DropFile form باستدعاء الـ DragAcceptFiles . لإلغاء السحب والاسقاط، يقوم الـ OnDestroy procedure باستدعاء نفس الـ function، ولكنه يمرر False للـ procedure الثانى.

تعرف الـ TMainForm class الـ handler الخاص برسالة الـ WMDropFiles. عندما يرسل الـ Windows رسالة الـ WMDropFiles للنافذة، تحتوى الـ parameter Msg على المعلومة التى تم تمريرها. يستخدم الـ procedure implementation هذه المعلومة فى استدعاء الـ WMDropFile للحصول على اسم الملف المسقط. ولان هذا الاسم null-terminated string يقوم البرنامج بتمريره للـ StrPas لتحويل النص الى Pascal string، ويتم تعينه للـ Caption الخاص بالـ FileNameLabel وتمريره لـ LoadFromFile method التابع لـ Memo1 . إننى أضيف استدعاءاً للـ LowerCase لتحويل اسم الملف الى الحروف الصغيرة، والتى هى أسهل على العين.

وهناك مشكلة واحدة مع هذه التقنيات وهى أنها لا تعمل عند سحب اسماء ملف على أيقونة التطبيق وهو فى حالة الـ minimized والسبب هو أن نافذة الـ form الرئيسية لـ Delphi تكون غير مرئية عند تصغير البرنامج. لتنفيذ السحب والإسقاط لهذه الحالة، يجب أن تضيف code للتطبيق. هذه هى النافذة التى توضح الأيقونة المصغرة- إنها ليست نفس النافذة المستخدمة من قبل الـ form.

اتبع هذه الخطوات لإقامة سحب وإسقاط لأيقونة مصغرة الى تطبيق الـ DropFiles . أضف استدعاءاً ثانياً للـ DragAcceptFiles فى OnCreate الخاص بالـ form . تخبر العبارة Windows أن نافذة الـ Application يمكن أن تقبل ملفات السحب والإسقاط :

```
DragAcceptFiles(Application.Handle, True);
```

لإبطال السحب والإسقاط لهذه النافذة، أضف عبارة مشابهة لـ OnDestroy للـ form، ولكن قم بتمرير False لـ DragAcceptFiles الـ parameter الثانى :

```
DragAcceptFiles(Application.Handle, False);
```

لا يتعامل الـ Application object مع الرسائل بنفس طريقته مع الـ form class . تصل الرسائل فى حالة أولية، بالضبط كما تم تمريرها بواسطة الـ Windows، وإنك تحتاج الى procedure خاصاً للتعامل معها . قم بتعريف الـ procedure كما يلى (اسمه لا يهم) فى القطاع protected للـ form class . على سبيل المثال، أضف هذا التعريف تحت ذلك الخاص بالـ WMDropFiles :

```
procedure AppOnMsg(var Msg: TMsg; var Handled:
 Boolean);
```

يعتبر الـ Parameter Msg من نوع الـ TMsg، والذى يتم تعريفه بواسطة Delphi فى الـ Windows.pas بـ :

```
tagMSG = packed record
hwnd: HWnd;
  message: UINT;
    wParam: WPARAM;
    lParam: LPARAM;
    time: DWORD;
    pt: TPoint;
end;
TMsg = tagMSG;
```

إنه فى هذه الـ form تصل الرسائل فى الـ Application . فى هذه الحالة ، نحن نهتم برسالة واحدة فقط ، وهى wm_DropFiles . بالإضافة الى ذلك ، نريد أن نميز أن هذه الرسالة فقط عند تصغير التطبيق . توضح القائمة (١١-٧) كيفية كتابة الـ handler .

يقوم الـ AppOnMsg procedure بفحص الـ Message.Msg لرسالة الـ wm_DropFiles . اذ كان التطبيق قد تم تصغيره أيضاً أيضاً (يقوم الـ IsIconic بإرجاع True) ، يستدعى الـ DragQueryFile procedure للحصول على اسم ملف وتحميل نصه فى الـ Memo object للبرنامج .

القائمة (١١-٧) : أدخل برنامج الـ handler هذا فى DropFiles لتشغيل ملفات السحب والإسقاط للأيقونة المصغرة للتطبيق

```
procedure TMainForm.AppOnMsg(var Msg: TMsg; var
Handled: Boolean);
var
 Filename: array[0 .. 256] of Char;
begin
 with Application do
   if (Msg.Message = wm_DropFiles) and IsIconic(Handle)
  then
   begin
    DragQueryFile(THandle(Msg.WParam),
       0, Filename, SizeOf(Filename));
      with FileNameLabel do
      begin
     Caption := LowerCase(FileName);
         Memo1.Lines.LoadfromFile(Caption);
        end;
      DragFinish(THandle(Msg.WParam));
     end;
end;
```

هناك خطوة إخيرة تجعل هذا الـ code يعمل بشكل جيد . يجب أن تعين الـ handler (وهو AppOnMsg فى هذا المثـال) للـ OnMessage الخـاص بالـ

Application . لضـمـان أن البـرنامج يتلقى كل الرسـائل المسـتـهـدفـة ، افـعل هذه الخطوة مـبكراً- فى الـ OnCreate للـ form ، مـثـلاً . قـم بإنشـاء برنامج للـ OnCreate الخاص بالـ DropFile ، وأدخل هذه العبارة :

```
Application.OnMessage := AppOnMsg;
```

يمكنك الآن تشغيل الـ DropFile وتصغير نافذته على الحاسب المكتبى للـ Windows . اسحب اسم ملف نص من الـ File Manager واسقطه على الأيقونة . عندما تفتح نافذة الـ DropFile ، سوف ترى نص الملف فى الـ Memo object .

قراءة وكتابة ملفات الـ ini. :

تبسط الـ TIniFile class الخاصة بـ Delphi العمل الشاق الخاص بقراءة وكتابة ملفات بدء الـ Windows المعرفة بالامتداد الـini. . لا الـ TIniFile class لا تعتبر component ، وهى لا تظهر فى لوحة الـ VCL لـ Delphi . لهذا السبب ، يجب عليك إنشاء وتحرير الـ TIniFile objects ، واستـدعاء methods فى ظل تحكم البرنامج كاملاً .

فى غالبية البرامج ، تحتاج الى two procedures لقراءة وكتابة مواصفات ملف الـini. . على سبيل المثال ، يمكنك حفظ خيارات ومواصفات اخرى ، التى يستطيع المستخدمون الخبراء تعديلها بتحميل ملف الـini. فى text editor . لإنشاء وتحديث الملف ، إنشئ اولاً TIniFile objects . قم بتعريفه كما يلى :

```
var
  IniFile: TIniFile;
```

ثم ، فى قلب الـ procedure ، إنشئ IniFile object بإستدعاء الـ Create method الخاص بالـ TIniFile :

```
IniFile := TIniFile.Create('YourProg.ini');
```

عندما تنتهى من استخدام الـ TIniFile ، حدده بهذه العبارة :

```
IniFile.Free;
```

اذا كنت لم تذكر معلومات عن اسم المسار والـ drive لمتغير اسم الملف ، يتم إنشاء ملف الـ ini.فى دليل الـ Windows . هذا هو الأفضل دائماً ، ولكن بعض

المبرمجين ينشئون ملفات الـ ini.فى دليل الخاص بالتطبيق. اينما تضع الملف، بعد إنشاءالـ IniFile object ، استدع الـ method لقراءة وكتابة مواصفات البدء. يجب ان يكون كل صفه فى القطاع المحاط بالاقواس، متبوعاً بقيمة واحدة أو أكثر. على سبيل المثال، ها هى الاسطر الثلاثة الأولى من ملف الـ SysColor.ini، الذى تم إنشاءه بتطبيق الـ SysColor الموضح بعد ذلك فى هذا الفصل:

```
[SysColor]
Scroll Bar=12632256
Background=4210688
```

يمكنك قراءة مواصفات ملف الـini. فى الـ TStringList object أو الـ TStrings بطريقتين. عرف الـ object كما يلى:

```
var
  StringList: TStringList;
```

إنشىء قائمة الـ string واستدع الـ ReadSectionValues لـ IniFile object لتحميل القائمة بالإضافة إلى تنفيذ مواصفات الملف. على سبيل المثال، استخدم عبارات مثل هذه:

```
StringList := TStringList.Create;
IniFile.ReadSectionValues('SysColor', StringList);
```

تقوم العبارة الأولى بإنشاء قائمة الـ string وتستدعى الثانية الـ ReadSectionValues التى تمرر اسم القطاع [SysColor] بلا اقواس كـالـ string أو الـ StringList object. يتضمن كل عنصر فى قائمة الـ string إسم العنصر، علامة يساوى، وقيمة، بلا مسافات حول علامة يساوى. بالتبادل، لتحميل اسماء العناصر فقط، إستدع الـ ReadSection هكذا:

```
IniFile.ReadSection('SysColor', StringList);
```

يوفر الـIniFile ايضاً methods لقراءة وكتابة قيم منفردة. على سبيل المثال، يقوم الـSysColor بكتابة مواصفات الـ CheckBox بالعبارة:

```
IniFile.WriteBool('Options', 'Save settings',
  SaveCheckBox.Checked);
```

الـ argument الاولى، 'Options' ، تحـــدد القطاع الذى يذهب إليـــه العنصر. والقيمة المتغيرة الثانية هى إسم هذا العنصر، والاخيرة هى قيمته. اذا كانت الـ Checked محددة بـ True، سوف تبدو سطور ملف الـ ini. الناتج هكذا:

```
[Options]
Save settings=1
```

لقراءة المواصفات، استدع احدىRead methods الخاصة بالـ IniFile class. على سبيل المثال، يقوم الـSysColor بتحميل قيمة الـ CheckBox لتحديد الـSave بالعبارة:

```
SaveCheckBox.Checked :=
  IniFile.ReadBool('Options', 'Save settings', False);
```

تعمل methods القـراءة والكتـابة الأخـرى- مـثل الـReadInteger، ReadString، WriteInteger، وWriteString - بطريقة مشابهة، ولكن إقرأ واكتب قيم للأنواع المشار إليها.

بعـد قـراءة مـواصـفـات ملف الـ ini. فى قـائمـة الـ string يمكنك وضع مواصفات معينة. افعل هذا بفهرسة الـ Values array لقائمة لـ string المستهدفة. على سبيل المثال، اذا كـانت الـ T هى TStringList object تم تحـميله بواسطة الـ ReadSectionValues method لـ IniFile object، إذن الـValues[S] تساوى قيمة الـ string للعنصر المعرف بـS. يمكنك تعيين هذه القيمة لـ string:

```
S := T.Values['Button color'];
```

ولكن، إنك تريد غالباً ان تحولها قيمة عدد صحيح أو أى قيمة ثنائية اخرى للاستخدام فى البرنامج. افعل هذا باستدعاء function تقوم بمثل هذا مثل الـ StrToImt:

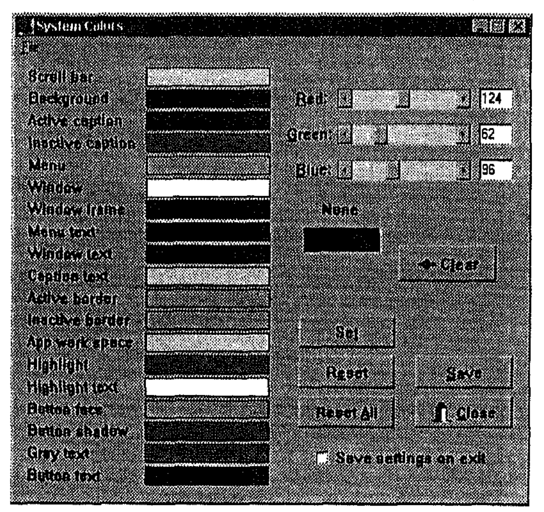
```
K := StrToInt(T.Values['Button color']);
```

SysColor Utility :



على القرص المدمج: ان البرنامج والطول الكامل، SysColor، يضع هذه المفاهيم موضع التنفيذ. ان الـ source code طويلة الى حد ما

بحيث يصعب ذكرها هنا كاملة ، لذا ساقدم اجزاء منها فى حينه . ويتبع التوضيح ، بالطبع ، كل جزئية قائمة . توجد الـ source code الكاملة ، بالطبع ، على القرص المدمج فى دليل الـ Source\SysColor . يوضح شكل (١١-٥) البرنامج راجع الفقرة " كيفية استخدام الـ SysColor " لمعرفة التعليمات .



شكل (١١-٥): يعرض الـ SysColor ويمكنك من اختيار ١٩ لون لنظام الـ Windows، مثل لون خلفية النوافذ، ولون عنوان النافذة الـ active و الـ inactive

كيفية استخدام الـ SysColor

بما انك درست الـ SysColor، فانت مستعد لاستخدامه لاحظ ان المستطيلات الملونة تعرض مواصفات اللون الحالى. اضغط أى مربع لون لاختياره وتنقل لونه الى مربع العينة الكبير، الذى يحمل عنوان None. يظهر المربع اللون المختار واسمه. اضغط Click لفك ارتباط مربع العينة بأى لون مختار (هذا جيد فى اختيار قيم الالوان دون التأثير على صفة الـ labeled).

استخدم الـ Red scroll bars، Green و Blue لتعديل قيم الالوان لـ Windows. على سبيل المثال، غير تعليق الـ Active للون الاحمر النادى واضغط Set . انك ترى تعليق النافذة الـ active يتغير لخلفية حمراء.

اضغط زر Reset لإعادة الالوان المحددة فى المواصفات الاصلية عندما تبدأ البرنامج. إتبع هذا بضغط Set اذا كنت تريد إعادة الوان نظام الـ Windows الى قيمتها الاصلية.

"تابع"

اضغط زر الـ Reset All لإعادة تحديد الالوان المحددة بقيم البدء الخاصة بها وكذلك استعادة الوان نظام الـ Windows. يؤدى هذا الزر نفس الأعمال التى تتم عند ضغط الـ Reset متبوعاً بالـ Set.

اضغط زر الـ Save لحفظ اختيارات اللون الحالية فى الـ SysColor.ini، الموجود فى دليل الـ Windows. فى المرة التالية التى تقوم فيها بتشغيل الـ SysColor، فهو يعرض القيم المحفوظة فى مستطيلات محددة. اذا لم تكن تريد استخدام الوان محفوظة، احذف الـ SysColor.ini وادخل البرنامج، الذى يعرض قيم لون الـ Windows الحالية.

اضغط زر Close لإنهاء البرنامج (أو يمكنك اختيار أمر الـ File|Exit). اذا كانت مواصفات الـ Save على الـ check box الخاص باختيار الخروج قد تم تشغيلها، يقوم البرنامج بتحديث الـ SysColor.ini عندما تتركه. إجعل check box الخاص بالاختيار disable اذا لم تكن تريد حفظ اختيارات لونك تلقائياً.

توضح القائمة (١١-٨) قطاع الـ interface لـ SysColor . ان الـLabel والـ Shape المرقمة تعرض مستطيلات اللون المحددة فى النافذة. (يقدم الباب الثالث عشر الـ Shape وتقنيات جرافيك اخرى .

القائمة (١١-٨): (قطاع interface) SysColor\Main.pas

```
unit Main;

interface

uses
  SysUtils, Windows, Messages, Classes, Graphics,
     Controls, Forms, Dialogs, StdCtrls, Buttons, ExtCtrls,
     AboutDlg, IniFiles, Menus;

const
 maxColors = 19;         { Number of Windows system colors }
    redMask = $000000FF;          { Red value extraction mask
   }
```



```
greenMask = $0000FF00;       { Green value extraction mask }
 blueMask = $00FF0000;        { Blue value extraction mask
}
 iniFileName = 'SysColor.ini'; { In the Windows directory
}


type
 TMainForm = class(TForm)
    BlueEdit: TEdit;
      BlueLabel: TLabel;
      BlueSB: TScrollBar;
      ClearBitBtn: TBitBtn;
      CloseBitBtn: TBitBtn;
      ColorEdit: TEdit;
      ColorLabel: TLabel;
      FileAbout: TMenuItem;
      FileExit: TMenuItem;
      FileMenu: TMenuItem;
      GreenEdit: TEdit;
      GreenLabel: TLabel;
      GreenSB: TScrollBar;
      Label1: TLabel;
      Label2: TLabel;
      Label3: TLabel;
      Label4: TLabel;
      Label5: TLabel;
      Label6: TLabel;
      Label7: TLabel;
      Label8: TLabel;
      Label9: TLabel;
      Label10: TLabel;
      Label11: TLabel;
      Label12: TLabel;
      Label13: TLabel;
```

```
Label14: TLabel;
  Label15: TLabel;
  Label16: TLabel;
  Label17: TLabel;
  Label18: TLabel;
  Label19: TLabel;
  MainMenu1: TMainMenu;
  N1: TMenuItem;
  RedEdit: TEdit;
  RedLabel: TLabel;
  RedSB: TScrollBar;
  ResetAllBitBtn: TBitBtn;
  ResetBitBtn: TBitBtn;
  SaveBitBtn: TBitBtn;
  SaveCheckBox: TCheckBox;
  SetBitBtn: TBitBtn;
  Shape1: TShape;
  Shape2: TShape;
  Shape3: TShape;
  Shape4: TShape;
  Shape5: TShape;
  Shape6: TShape;
  Shape7: TShape;
  Shape8: TShape;
(continued)
Listing 11-8: (continued)
  Shape9: TShape;
  Shape10: TShape;
  Shape11: TShape;
  Shape12: TShape;
  Shape13: TShape;
  Shape14: TShape;
  Shape15: TShape;
```

```
Shape16: TShape;
  Shape17: TShape;
  Shape18: TShape;
  Shape19: TShape;
  procedure ClearBitBtnClick(Sender: TObject);
  procedure EditChange(Sender: TObject);
  procedure FileAboutClick(Sender: TObject);
  procedure FileExitClick(Sender: TObject);
  procedure FormClose(Sender: TObject;
var Action: TCloseAction);
    procedure FormCreate(Sender: TObject);
  procedure FormDestroy(Sender: TObject);
  procedure ResetAllBitBtnClick(Sender: TObject);
  procedure ResetBitBtnClick(Sender: TObject);
  procedure SaveBitBtnClick(Sender: TObject);
  procedure SaveCheckBoxClick(Sender: TObject);
  procedure SBChange(Sender: TObject);
  procedure SetBitBtnClick(Sender: TObject);
  procedure ShapeMouseDown(Sender: TObject;
Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
     private
    IniFile: TIniFile;
  IniItemList: TStringList;
  EditControls: array[0 .. 2] of TEdit;
  ScrollBars: array[0 .. 2] of TScrollBar;
  Shapes: array[0 .. maxColors - 1] of TShape;
  CurrentShape: TShape;
  procedure UpdateColor;
  procedure InitSysColorArray;
  procedure ChangeSystemColors;
  procedure SetScrollBars(C: TColor);
  procedure LoadSettings;
  procedure SaveSettings;
```

```
public
  { Public declarations }
   end;

var
 MainForm: TMainForm;

implementation

{$R *.DFM}
```

يعرف البرنامج بعض الثوابت قرب بداية القائمة :

* الـ maxColors تساوى عدد الوان النظام بالـ Windows .

* والـredMask، greenMask ، وblueMask تعتبر اعداد صحيحة عشرية ذات ٣٢ بت تستخدم لاستخلاص قيم الوان منفردة من الـ TColor objects .

على سبيل المثال، التعبير C و greenMask يعزل قيمة اللون الاخضر للـ TColor . يعتبر الـ iniFileName هو اسم ملف بدء البرنامج . ولان الاسم لا يحتوى على حرف drive أو أى معلومة عن مساره، يكون موضعه هو دليل الـ Windows .

الـ TMainForm class تعرفobjects متعددة والتى تنشئ واجهة تطبيق النافذة . يتماشى رقم كل Label مع الـ Shape المقابل (على سبيل المثال، الـ Label7 يذهب مع الـ Shape7) . وهذه العلاقة ليست عشوائية ، وكما سأوضح بعد قليل ، عندما يكون لديك العديد من الـ components المتعلقة ببعضها ، يمكنك الاستفادة من اعراف تسمية الـ objects بصورة تلقائية الخاصة بـ Delphi لكتابة code لـ objects متعلقة ببعضها . تعتبر غالبية الـ objects والـ event handlers الاخرى واضحة، لذا سوف اشرح فقط العناصر التى اضفتها للقطاع private للـTMainForm . لقد تم إضافة ستة متغيرات وهى :

• **الـIniFile:** هذا الـobject تم إنشاؤه فى OnCreate الخاص بالـ form وتم تدميره فى الـ OnDestroy. . توفر الـ IniFile وصولاً الى ملف الـ Syscolor.ini

لـ procedures متنوعة . يفتح الـ object هذا الملف ويغلقه على حسب الحاجة لقراءة وكتابة مواصفاته ، لذا فمن الأكثر فعالية ان تعرف الـ object فى الـ form class عن ان يعرف الـ TIniFile objects فى procedures منفردة . وكذلك ، اذا تم تعريفه محلياً لـ procedures ، فإن ملف الـ ini. سيتم فتحه واغلاقه كل مرة يتم فيها استدعاء الـ procedures .

● **الـ IniItemList**: هذه هى قائمة لتعريف ملف الـ SysColor.ini، والتى تخزن مواصفات اللون على انها اعداد عشرية صحيحة ذات ٣٢ بت . على سبيل المثال ، ان مدخل الـ IniItemList الخاص بالـ SysColor.ini يحدد 4210688-Background هو الـ 'Background' string . لقد وجدت انه من المساعدة ان نبقى العناصر المحددة فى قائمة string منفصلة ، والتى سوف أوضح المزيد عنها فى هذا الباب .

● **الـ EditControls**: هذا الـ array يسهل الوصول الى نوافذ الـ Edit الثلاث الواقعة بعد الـ scrollbars . (يستخدم تطبيق الـ ColorDlg بالباب العاشر ايضاً هذا الـ array) .

● **الـ Shapes**: يحتوى هذا الـ array على مراجع لـ ١٩ Shape objects تعرض الوان نظام الـ Windows . ان استخدام array يبسط الـ code - فمثلاً ، بدلاً من كتابة ١٩ handlers ، يمكن للبرنامج ان يستخدم الـ array للإشارة الى مستطيل لون n بالتعبير [Shapes[n . كما يوضح الـ SysColors ، الـ Reference object array ، والتى سوف تجدها مفيدة فى كثير من الحالات .

● **الـ ScrollBar**: هو array يسهل الوصول للـ scrollbar object الثلاث . (تطبيق الـ ColorDlg للباب العاشر يستخدم هذا الـ array) .

● **CurrentShape**: هذا هو مرجع لمستطيل اللون الذى تم اختياره حالياً . عندما يساوى nil ، لا يتم اختيار أى شئ . بالتبادل ، كان يمكن ان استخدم فهرس عدد صحيح فى الـ Shapes array للإشارة للون الحالى ، ربما 1- دون تمثيل أى اختيار . ولكن ، لان كل Delphi objects تعتبر Reference ، فمن الأفضل تعريف المتغيرات مثل الـ CurrentShape االتى تشير الى الـ objects المستهدف

فـعليـاً. على سـبـيل المثـال، يسـتطيع البـرنامج تعـيين Shapes[n] للـ CurrentShape، ثم يستخدم الـ CurrentShape لأداء اعمال على الـ objects المختار.

بالإضافة الى المتغيرات الست السابقة فإن الـ SysColor، يعرف ستة procedures خاصة فى الـ TMainForm class. فيما يلى تقديم مختصر لكل procedure- ووصف كامل عن كيفية عملهم:

- **UpdateColor**: يحدد هذا الـ procedures مربع اللون الكبير، الـ Edit object ويسمى الـ ColorEdit، للون الذى يمثله الـ ScrollBars الثلاث. يحدد الـ procedures ايضاً لون الـ CurrentShape اذا كان هناك واحداً مختاراً. وكذلك يقوم الـ UpdateColor بحفظ اللون الحالى فى الـ SysColorArray array العام.
- **InitSysColorArray**: هذا الـ procedure ينشئ الـ SysColorArray array، الذى يخزن الوان نظام الـ Windows الحالية مع مجموعة اللون المعدلة حالياً. ان الـ array يجعل من السهل على البرنامج ان يعيد تحديد الالوان لقيم البدء الخاصة بها. يستخدم البرنامج ايضاً الـ SysColorArray لتحديث ملف الـ SysColor.ini.
- **ChangeSystemColors**: هذا الـ procedures يحـول الى الـ Windows اختيارات اللون الحالى. لكل لون، تتلقى كل التطبيقات رسالة wm_SysColorChange والتى تشير الى ان التطبيق يجب ان يقوم بتحديث نوافذه، ازراره والعناصر المؤثرة الاخرى. تقوم تطبيقات Delphi بهذه التحديثات بصورة تلقائية لا يجب عليك كتابة code لدعم رسائل الـ wm_SysColorChange.
- **SetScrollBars**: هذا الـ procedures يحدد خصائص الـ properties للـ ScrollBars الثلاث بقيم تتماشى مع لون يتم تمريره كـ argument.
- **LoadSettings**: هذا الـ procedures يقرأ ملف الـ SysColor.ini، ان وجد.

- **SaveSettings** : هذا الـ procedures ينشئ أو يحدث مواصفات الألوان فى ملف الـ SysColor.ini .

فهم procedures البدء والتعريفات الـ global للـ SysColor.ini:

توضح القائمة (١١-٩) procedures البدء والتعريفات الـ global للـ SysColor.ini . الشرح يعقب القائمة .

القائمة (١١-٩): SysColor\Main.pas (بدء وتعريفات)

```
type
  { Holds current system colors }
   SysColorRec = record
     OriginalColor: TColor;  { Color on starting program }
     CurrentColor: TColor;   { New color selected by user
   }
    end;

var
{ Array of SysColorRec values }
 SysColorArray: array[0 .. maxColors - 1 ] of SysColorRec;

{ Update sample colors from scrollbar positions }
procedure TMainForm.UpdateColor;
begin
{ Update color edit box (the large sample window) }
 ColorEdit.Color :=
     RGB(RedSB.Position, GreenSB.Position, BlueSB.Position);
{ Update labeled Shape color box and SysColorArray }
 if CurrentShape <> nil then with CurrentShape do
    begin
     Brush.Color := ColorEdit.Color;
        SysColorArray[Tag - 1].CurrentColor := Brush.Color;
      end;
end;
```

```
{ Load system colors into the SysColor array }
procedure TMainForm.initSysColorArray;
var
 I: Integer;
begin
 for I := 0 to maxColors - 1 do with SysColorArray[I] do
    begin
      OriginalColor := GetSysColor(I);
        CurrentColor := OriginalColor;
        Shapes[I].Brush.Color := OriginalColor;
      end;
end;

{ Change system colors to values in SysColorArray }
procedure TMainForm.ChangeSystemColors;
var
 I: Integer;
    InxArray: Array[0 .. maxColors - 1] of Integer;
    ClrArray: Array[0 .. maxColors - 1] of TColor;
begin
    for I := 0 to maxColors -1 do
    begin
      InxArray[I] := I;
        ClrArray[I] := SysColorArray[I].CurrentColor;
      end;
    SetSysColors(maxColors, InxArray[0], ClrArray[0]);
end;
```

يقوم نوع بيانات الـ SysColorRec بتعريف سجل من قيمتي TColor . تعتبر الـ OriginalColor هي قيمة اللون المأخوذة من الـ Windows عند البدء. تعد الـ CurrentColor هي قيمة اللون الموضحة في مستطيل اللون المحدد. يحمل الـ SysColorArray يحمل ١٩ SysColorRec Record ، واحداً لكل لون في نظام Windows .

يقوم الـ UpdateColor Procedure باستدعاء الـ Windows RGB function لدمج خصائص الـ Position للـ ScrollBar الثلاث فى TColor-compatible LongInt ذى الـ ٣٢ بت . ان تعيين هذه القيمة لخاصية الـColor لـ ColorEdit object يؤدى الى تحديث عينة اللون الكبيرة تحت الـ Scrollbars.

اذا اختار المستخدم لون محدد (الـ CurrentShape لا يساوى nil)، تقوم الـ UpdateColor ايضاً بتعيين اللون الجديد للخاصية الفرعية Brush.Color للـ CurrentShape ، ويحفظ اللون فى الـ SysColorArray . لاحظ ان الـ Tag الخاص بالـ Shape يساوى رقم الـ Index التابع له فى الـ SysColorArray .

يوضح الـ Procedure InitSysColorArray كيفية الحصول على تحديدات اللون الحالى من الـ Windows . تقوم الـ for loop باستدعاء الـ API GetSysColor function مع الـ argument I مساوياً لـ Index اللون . تحفظ الـ loop كل لون فى SysColorArray، وتعرضه بواسطة تحديث القيمة الفرعية للـ Brush.Color للـ Shapes . هذا يعتبر نموذجاً جيداً على مدى فائدة الـ arrays المشيرة للـobjects مثل الـ Shapes .

يحول الـ ChangeSystemColors Procedur قيم اللون الـ ١٩ للـ Windows باستدعاء الـ API function SetSysColors . تتطلب هذه function ثلاثة arguments : عدد الالوان، array للاعداد الصحيحة لـ Index اللون، وarray لقيم اللون ذات الـ ٣٢ بت . والعبارة التى تستدعى الـ SetSysColors تمرر عناوين الـ arraysوالـ InxArray والـ ClrArray، بفهرسة عناصرهم الأولى .

انظر الـ SetSysColors فى مرجع الـ API online الخاصة بـ Delphi . الـ two arrays تم تعريفهما على انهما var parameters غير مكتوبة . لإعطاء عنوان array لهذا الـ parameter ، قم بتمرير عنصره الأول، كما يوضح الـ procedure ChangeSystemColors.

استخدام الـsysColor's form - maintence event handlers:

توضح القائمة (١١-١٠) ثلاثة من الـ handler للحفاظ على الـform، والتى تعد مسئولة عن بدء الـ SysColor objects ولأداء اعمال المسح عندما ينتهى البرنامج . التوضيح يعقب القائمة .

يتعامل الـ FormCreate Procedure مع الـ OnCreate الخاصة بالـ form.
يبدء الـ procedure بمتغيرات ويستدعى methods لإعادة الـ display. توضح الـ for loop الأولى تقنية مفيدة فى تكوين و object Reference array ولنفحصها بعمق :

```
for I := 0 to maxColors -1 do
  Shapes[I] :=
      TShape(FindComponent('Shape' + IntToStr(I + 1)));
```

القائمة (١١-١٠): SysColor\Main.pas (يوضح الـ handlers)

```
{ Initialize TObject control arrays }
procedure TMainForm.FormCreate(Sender: TObject);
var
 I: Integer;
begin
{ Construct IniFile object instance }
 IniFile := TIniFile.Create(iniFileName);
{ Miscellaneous initializations }
 CurrentShape := nil;  { No selected labeled color shape }
{ Assign object references to easy-access arrays }
 EditControls[0] := RedEdit;
  EditControls[1] := GreenEdit;
  EditControls[2] := BlueEdit;
  ScrollBars[0] := RedSB;
  ScrollBars[1] := GreenSB;
  ScrollBars[2] := BlueSB;
{ Assign Shape color box object references to Shapes array }
 for I := 0 to maxColors -1 do
    Shapes[I] :=
       TShape(FindComponent('Shape' + IntToStr(I + 1)));
{ Create item list from Labels for Ini file read/write }
 IniItemList := TStringList.Create;
  with IniItemList do
  for I := 0 to maxColors -1 do
    Add(TLabel(FindComponent(
```

```
        'Label' + IntToStr(I + 1))).Caption);
  { Load current colors and possible Ini file settings }
   InitSysColorArray;  { Initialize SysColor array }
    LoadSettings;      { Load SysColor.ini settings if present }
  end;

  { Save settings if SaveCheckBox selected }
  procedure TMainForm.FormClose(Sender: TObject;
   var Action: TCloseAction);
  begin
     if SaveCheckBox.Checked then
      SaveSettings;  { Save settings in SysColor.ini }
  end;

  { Last chance to clean up }
  procedure TMainForm.FormDestroy(Sender: TObject);
  begin
     IniItemList.Free;
     IniFile.Free;
  end;
```

ان الهدف هو تعيين reference للـ Shape لكل object من الـ Shape objects، كل مستطيل يمثل لون واحد محدد فى النافذة. تقوم الـ FindComponent Function بإرجاع الـ reference لأسم الـobject الذى تم تمريره كـ string . للإشارة الى Shape1، Shape2، . . . ، Shape19 ، يستخدم البرنامج الـ IntToString لتحويل متغير control زائد واحد الـ loop if الى string، يكون ملحق بالـ Shape . لاتمام التعيين، يضع البرنامج قيمة إدخال الـ FindComponent فى TLabel object . يمكنك إيجاد استخدامات متعددة للـ FindComponent خاصة عند إنشاء arrays of object reference مثل الـ Shapes .

وتقوم for loop مشابهة فى نفس الـ procedure بإنشاء قائمة للمتغيرات لملف الـ SysColor.ini . بعد إنشاء الـ IniItemList ، وهو TStringList object، تقوم الـ loop بإدخال الـ Labels الخاصة بالبرنامج كمواصفات ملف الـ

ini. . وهنا مرة اخرى، يقوم الـ FindComponent بإرجاع reference objects للـ Label1، Label2، . . . ، Label19 ، يضع البرنامج قيمة إرجاع الـ function في الـ TLabel و يضع في الـ IniItemList حقل الـ Caption الخاص بهذا الـ object. اخيراً، يقوم الـ FormCreate باستدعاء الـ InitSysColorArray للحصول على مواصفات اللون الحالي للـ Windows . بعد ذلك، يقوم الـ LoadSettings بقراءة الـ SysColor.ini ، ان وجد .

يتعامل الـFormClose Procedure مع الـ OnClose للـ form. يستدعي البرنامج الـSaveSettings لتحديث أو إنشاء الـ SysColor.ini اذا قمت بتشغيل مواصفات الـSave عند خروج الـ CheckBox. يتعامل الـ FormDestroy Procedure مع الـ OnDestroy . هذه هي الفرصة الاخيرة للبرنامج للقيام بالمسح قبل الانتهاء .

كما هو موضح هنا، يعتبر هذا الـevent فرصة جيدة لتحرير الذاكرة . المخصصة للـ objects . في هذه الحالة، يحرر البرنامج الـ IniItemList object الذي انشأه OnCreate الخاص بالـ form .

التعامل مع الـ events للـ ScrollBar objects، والـ Edit، والـ Shape ،

توضح القائمة (١١-١١) كيف يتعامل الـ SysColor مع events للـ ScrollBar، الـ Edit ، و الـShape . الشرح يعقب القائمة .

القائمة (١١-١١): SysColor\Main.pas الـ SysColor event handlers ، الـ Edit ، والـShape

```
{ Set scrollbar positions to match color C }
procedure TMainForm.SetScrollBars(C: TColor);
begin
{ The following assignments also update Edit boxes }
 RedSB.Position   := C and redMask;
    GreenSB.Position := (C and greenMask) shr 8;
    BlueSB.Position  := (C and blueMask) shr 16;
end;

{ Update values in Edit boxes for ScrollBar changes }
procedure TMainForm.SBChange(Sender: TObject);
begin
```

```
    with Sender as TScrollBar do
      EditControls[Tag].Text := IntToStr(Position);
        UpdateColor;
  end;

  { Update scrollbar positions for Edit box changes }
  procedure TMainForm.EditChange(Sender: TObject);
  begin
      with Sender as TEdit do
        ScrollBars[Tag].Position := StrToInt(Text);
  end;

  { Select color shape on mouse down event }
  procedure TMainForm.ShapeMouseDown(Sender: TObject;
    Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
  var
    P: TLabel;   { Pointer to matching TLabel object }
  begin
    CurrentShape := TShape(Sender);  { Save clicked Shape }
      P := TLabel(FindComponent(
        'Label' + IntToStr(CurrentShape.Tag)));
          if P <> nil then
        ColorLabel.Caption := P.Caption;  { Show color name }
          SetScrollBars(CurrentShape.Brush.Color); {Synch scroll
    bars}
  end;
```

يقوم الـ SetScrollBars Procedure بتحديث خصائص الـ Position للـ ScrollBar الثلاثة للتطابق مع القيم الفرعية الحمراء والخضراء والزرقاء فى الـ C parameter الخاص بالـ TColor . ان تعيين قيم للـ Position ينقل thumb box (ويسمى ايضاً scroll box) . الى مكان قريب داخل مدى الـ color . وتستخدم ثلاثة تعبيرات and منطقية ثوابت الـ redMask، والـ greenMask ، والـ blueMask لعزل القيم الفرعية فى اللون C . ان عامل الـ shr (الـ shift يميناً) ينقل القيم المعزولة الى المواضع الاقل اهمية فى الاعداد الصحيحة الناتجة ، والتى تم تعيينها للـ Position .

يوضح الباب العاشر كيف تتعامل مع الـ SBChange procedures والـ EditChange handle مع الـOnChange events لـ ScrollBar والـ Edit . ان قيم الـ Tag الخاصة بالـ objects تساوى مواضعهم المفهرسة فى الـ arrays EditControls والـ ScrollBars المشيرة للـ objects . مع هذه الـ arrays ، يمكن ان تتشارك الـ Edit والـ ScrollBar فى الـ event handler ويتعاونوا ، بحيث اذا قمت بإدخال قيمة فى نافذة Edit ، يتتبعها الـ ScrollBar thumb box . بنفس الطريقة ، ان نقل الـ thumb box يؤدى الى تحديث قيمة نافذة الـ Edit .

يتولى الـ ShapeMouseDown Procedure عملية ضغط مستطيل اللون المحدد لان الـ Shape components لا تعتبر Windows controls قابلة للاختيار ، لذا ليس لها OnClick events . ولكن ، يمكنك تميز ضغط الفأرة لهذه الـ objects وغيرها من الـ objects الغير معروضة فى نوافذ بإنشاء OnMouseDown event handler مثل الـ ShapeMouseDown .

يعين الـ Procedure اولاً للـ CurrentShape مرجعاً للـ Shape المختار ، يتم تمريره الى الـ Procedure فى الـ Sender parameter . يستدعى البرنامج بعد ذلك الـ FindComponent ليضع الـ Label(n) ، حيث ان الـ n تساوى قيمة الـ Tag للـ Shape . ان استخدام الـ Tags مع الـ FindComponent بهذه الطريقة تعتبر وسيلة فعالة لإنشاء علاقات فيما بين الـobjects وللاستفادة من قواعد التسمية لـ Delphi. فى المعتاد ، يجب ان تسمى الـ objects طبقاً لاغراضها ، ولكن فى هذه الحالة ، من الاسهل ان تعمل مع الـ ٣٨ objects Labels و Shapes باستخدام اسماءها الافتراضية ، وهى Label1 ، Label2 ، . . . ، Shape19 ، Label19 ، Shape2 ، . . . ، Shape19 .

إنشاء وقراءة وتحديث ملف الـ SysColor.ini:

ان آخر جزئية فى الـ SysColor code فى القائمة (١١-١٢) توضح كيف ينشئ البرنامج ويقرأ و يحدث ملف الـ SysColor.ini . التوضيح يعقب القائمة .

يقوم الـ LoadSettings Procedure بقراءة الـ SysColor.ini اذا وجد الملف . ان دراسة هذا الـ code سطراً بسطر يزودك بمعلومات يمكنك استخدامها فى نظم التعامل مع بدء الملف الخاصة بك . لحمل مواصفات الملف ، يقوم الـ

LoadSettings بإنشـــاء object من الـ TStringList وهو الـ IniValueList . يســتــدعى البــرنامج بعــد ذلك الـ ReadSectionValues للـ TStringList object . وتشير argument 'SysColor' الى القطاع الذى يجب قراءته فى ملف الـ ini ، وهو فى هذه الحـالة ، [SysColor] . كل عنصـر فى الـ IniValueList الناتج يعتبر String فى الـ form :

Identifier=Value

القائمة (١١-٢): SysColor\Main.pas

```
{ Load colors and options from SysColor.ini if present }
procedure TMainForm.LoadSettings;
var
 IniValueList: TStringList;
   I: Integer;
begin
   IniValueList := TStringList.Create;
   try
   { [SysColor] settings }
    IniFile.ReadSectionValues('SysColor', IniValueList);
     for I := 0 to IniValueList.Count - 1 do
     with SysColorArray[I] do
     begin
      CurrentColor := StrToInt(
          IniValueList.Values[IniItemList[I]]);
          OriginalColor := CurrentColor;
          end;
   { [Options] settings }
    SaveCheckBox.Checked :=
      IniFile.ReadBool('Options', 'Save settings', False);
     finally
    IniValueList.Free;
     end;
   ChangeSystemColors;
   InitSysColorArray;
end;
```

```
{ Create or update SysColor.ini color settings only }
procedure TMainForm.SaveSettings;
var
I: Integer;
begin
{ [SysColor] settings }
    for I := 0 to IniItemList.Count - 1 do
      IniFile.WriteString('SysColor', IniItemList[I],
          IntToStr(SysColorArray[I].CurrentColor));
end;

{ Write check box setting to SysColor.ini }
procedure TMainForm.SaveCheckBoxClick(Sender: TObject);
begin
  IniFile.WriteBool('Options', 'Save settings',
      SaveCheckBox.Checked);
end;
```

للحصول على جزئية الـ Value لكل صفه، استخدم الـ Values array فى قائمة الـ IniValueList string . يستخدم الـ LoadSettings هذه التقنية لتعيين قيم لون للـ SysColorarray . اختبر هذا التعيين بدقة :

```
CurrentColor := StrToInt(
  IniValueList.Values[IniItemList[I]]);
```

ان التعبير IniItemList[I] يشير إلى واحدة من الـ labels التى تم إنشاءها فى الـ OnCreate الخاص بالـ form، (انظر الـ FormCreate procedure والعبارات التى تبـدأ من ...with IniItemList). وهذه الـ strings تمثل جـزئيـة الـ Identifier لتحـديد الـ Identifier=Value . استـخـدم الـ Identifier string كفهرس فى الـ Values array للحصول على الـ Value الخاص بهذا التحديد. يعتـبـر هذا ايضاً string، لذا لكى تعينه للـ CurrentColor، يقـوم البـرنامج باستدعاء الـStrToInt .

يستخدم الـ LoadSettings تقنية ملف ini. اخرى لقراءة مواصفات الـ Save عند خروج الـ CheckBox استدع الـ ReadBool لـ IniFile object،

مروراً بعنوان القطاع ('Options')، ويعرف المتغير بـ ('Save settings')، وقيمة افتراضية. اذا كان التحديد موجوداً، فإن الـ ReadBool يقوم بإدخاله. اذا لم يكن، فإنه يدخل القيمة الإفتراضية، المعينة هنا لخاصية الـ Checked التابعة لـ SaveCheckBox. استدع الـ functions الـ ReadInteger والـ ReadString المتشابهتين لقراءة قيم الـ string والعدد الصحيح.

بعد تحميل مواصفات ملف الـ ini.، يقوم لـ LoadSettings بتحرير الـ IniValueList. يقوم الـ procedure باستدعاء الـ ChangeSystemColors لنقل الالوان التى تم تحميلها الى الـ Windows. والاستدعاء الأخير للـ InitSysColorArray يعيد بدء الـ array بالحصول على تلك الالوان المحفوظة مرة اخرى من الـ Windows. هذا يضمن ان القيم الموجودة فى الـ array هى التى تم استخدامها فعلاً- إنها قد تختلف فى بعض الاحيان نتيجة الطريقة التى يختار الـ Windows بها الالوان المتجانسة بشكل وثيق اعتماداً على امكانيات الـdisplay driver.

يعتبر الـSaveSettings Procedure ابسط من الـ LoadSettings. تقوم for loop باستدعاء الـ WriteString method الخاص بالـ IniFile لكتابة مواصفات اللون الحالى. قم بتحديد عنوان قطاع ملف الـ ini بـ ('SysColor')، وجزئية الـIdentifier لكل صفة بـ(InitItemList[I])، و string يمثل جزئية الـ Value الى الـ WriteString. وهنا، يقوم الـ IntToStr بتحويل قيمة الـ CurrentColor من الـ SysColorArray. عندما تنتهى من كتابة كل الألوان، يقوم الـSaveSettings بتحرير الـ IniFile object.

لاحظ ان الـ WriteString لا يقوم بحفظ خاصية الـChecked للـ SaveCheckBox. عندما حاولت ان افعل هذا بالضبط فى نسخة سابقة، لاحظت ان حفظ هذا الـ CheckBox قد كون حاله catch-22. يمكنك فحص الـ box، الذى تم حفظه فى ملف الـ ini، ولكن اذا لم تفحصه، لا يتم تحديث الملف، مما يؤدى إلى تحديث الملف مرة اخرى من الملف القديم عندما يبدأ الـ SysColor فى المرة القادمة.

والحل هو ان يقوم بتحديث قيمة الـ SaveCheckBox فى كل مرة يتغير فيها الـ CheckBox. يتولى الـ SaveCheckBoxClick Procedure هذه المهمة للـ

الخـاص بالـ control يقـوم الـ procedure باسـتـدعـاء الـ OnClick event WriteBool لحفظ الصفة الحالية (الذى يكون قد تغيرت بالفعل بمجرد استدعاء الـOnClick) . هذا يشير الى قاعدة هامة فى استخدام ملفات الـ ini. لحفظ خيارات برنامج- إنك قد تحتاج الى تحديث عناصر منفردة اذا كانت قيمهم تؤثر على اذا ما كان الـini. قد تم تحديثه .

اختبار ملف SysColor.ini:

توضح القائمة (١١-١٣) ملف SysColor.ini . ارجع الى هذه القائمة اثناء فحص نظم معالجة ملف الـini. الخاص بالـSysColor .

ان باقى قائمة الـ SysColor، والتى يمكن ان تجدها على القرص المدمج فى دليل الـ Source\SysColor ، يوفـر الـ event handler لأزرار النافـذة والـ CheckBox. وهذه الـ event handler، والتى لهـا اهداف واضـحـة، تقـوم باستدعاء الـ procedures اخرى فى البرنامج، لذا لن اذكرها أو اشرحها هنا . إننى اضفت تعليقات الى النافذة، فى حالة ان كنت تريد فحص المزيد من البرنامج .

القائمة (١١-١٣): ملف الـ SysColor.ini

```
[SysColor]
Scroll Bar=12632256
Background=4210688
Active caption=8755968
Inactive caption=8421440
Menu=12632256
Window=16777215
Window frame=0
Menu text=0
Window text=0
Caption text=16777215
Active border=4227072
Inactive border=8421440
App work space=12632256
Highlight=32768
Highlight text=16777215
```

```
Button face=12632256
Button shadow=8421504
Gray text=8421504
Button text=0

[Options]
Save settings=0
```

افكار للمستخدم الخبير

● اذا قمت بتخزين ملفات ini. فى مسارات محددة، تأكد من منح المستخدمين فرصة اختيار موضع بديل. فى اغلب الحالات، من الأفضل ان تخزن ملفات الـ ini فى دليل Windows، ولكن هذا ليس طلب. لتخزين ملفات الـ ini. فى دليل البرنامج المركب، استخدم ParamStr(0) function للحصول على مسار التطبيق.

● استدع الـ DiskFree لتحديد حجم مساحة القرص المتاحة، والتى قد تريد إضافتها الى directory اختيار الملف الخاص بهذا الباب. اختبر functions والدليل والملف الاخرى بالبحث فى online help الخاصة بـDelphi لـ".File-management routines

● لا تندفع وراء تقنيات التعامل مع الرسالة الخاصة بهذا الباب، قبل كتابة الـ message event handler لتطبيق أو form، افحص بدقة اذا ما كانت الإمكانية التى تحتاجها توجد بالفعل فى صورة event handler.

● فى الـ C، يمكن ان تقوم عبارة بتعيين قيمة ذات علامة مثل الـ 1- لـ unsignedWORD parameters، ان قواعد type-checking الخاصة بـ Pascal لا تسمح بهذا العمل، ولكن الغير المباشر أمر بسيط. على سبيل المثال، لتمرير 1- الى WORD parameter للـ Windows API Function استخدم الثابت العشرى المساوى وهو $FFFF.

● تذكر دائماً ان تقرير الـ objects التى تنشئها. عند إنشاء objects ديناميكية عامة مثل الـ TStringLists، قم بانشاءها فى الـ OnCreate الخاص بالـ

form . اسـتـدع Free لكل objects تم إنشـــاؤه فى الـ OnDestroy الخـاص بالـ form .

● لا تنشئ ملفـات مـثل الـ system.ini أو الـ win.ini ، اللذان لا يجب إنشـاء ملفات مشابهة لهما ولقد تم استبدالها بالـ Windows registry .

المشروعات التى يمكنك تجربتها

(١١-١) : اكتب ini file editor. .

(١١-٢) : اضف قطاعـاً إلى ملف الـ SysColor.ini ليـحـفظ وضع نافـذة البرنامج . ان تشغيل الـ SysColor يجب ان يعيد النافذة الى اخر وضع لها .

(١١-٣) : اكـتـب wildcard filter editor خـاص بك ، مـشـابهـاً لذلك الخـاص بـ Delphi ، واضـفـه الى directory dialog القـابل للتغيير الخاص بهذا الباب . اكتب user filters لإنشـاء ملف (.ini) .

(١١-٤) : اكتب file utility يمكن ان تغير مميزات الملف مثل مواصفات الـ Archive والـ ReadOnly . اســتـخــدم الـ function s . الـ FileGetAttr والـFileSetAttr فى نظـم إدارة الملف الخـاصـة بـ Delphi (ابحث عنهم فى الـ online) .

(١١-٥) : متقدم فى هذا الباب العديد من التقنيات اللازمة لإنشـاء file Explorer utility مخصصة . هناك دائماً سوقاً جيدة للبرامج من هذه النوعـيـة التى تدعم ادوات الـ Windows . المعـيـارية . كمشروع متقدم ، فكر فى كتابة البديل الخاص بك لـ Explorer utility الخـاصـة بالـ Windows . يمكنك مـثـلاً اضافة خـاصيـة تحافظ على drives القرص المتعددة مفتوحة- شئ لا يستطيع الـ Explorer ان يفعله .

(١١-٦) : متقدم - يقوم الـ SysColor بتحديث كل الوان نظام الـ Windows ، حتى اذا كانت قيم اللون لم تتغير . هذا ينتج عنه العديد من الرسائل التى تصدر صوتاً بين حواجز event التطبيق . أكتب نسخة أكثر فعالية تقوم بتحديث أقل عدد فقط من الالوان عندما يضغط المستخدمون زر الـSet .

ملخص:

- استخدم directory components والملف الموضحة فى هذا الباب فقط . عندما تحتاج File-Selection dialog مخصص . يوضح الباب التالى كيفية استخدام الـ components الـ SaveDialog والـ OpenDialog الخاصة بـ Delphi ، واللذان يعتبران كافيتين لـلـ directory interfac objects والملف .

- استخدم الـ DirectoryListBox ، والـ DriveComboBox ، والـ FileListBox ، والـcomponents FilterComboBox . لإنشاء File-Selection dialogs مخصصة . بربط هذه الـobjects ، فإنك تمكنها من العمل مع بعضها بحيث ، مثلاً ، عندما يقوم المستخدمون بتغيير الـ directories فى الـ DirectoryListBox ، يقوم الـ FileListBox المرتبطة به بصورة تلقائية بتحديث محتوياته .

- اضف ShellAPI لأمر الـ uses الخاص بالـ module . واستدع الـ ShellExecute لتشغيل برامج أو تحميل registered files. على سبيل المثال ، يمكنك عرض ملف نص فى الـ Windows Notepad utility بفتح ملف txt . بالـShellExecute .

- ان Delphi components لا تميز اسماء ملفات السحب والاسقاط ، ولكن يمكنك بسهولة إضافة هذه الامكانية بكتابة procedure يستجيب لرسالة الـ wm_DropFiles . لتشغيل ملفات السحب والاسقاط ، استدع الـ DragAcceptFiles فى الـ OnCreate بالـform .

- عندما يحدث minimize لتطبيق، تختفى الـform الرئيسية. لهذا السبب، فإن تنفيذ ملفات السحب والاسقاط لبرنامج minimized مثل ايقونة مما يتطلب من الـ Application object ان يستجيب لرسالة الـ wm_DropFiles. لبرمجة استجابة لهذه الرسالة أو لأى من رسائل الـ Windows الأخرى، قم بتعيين message handler لخاصية الـ OnMessage التابعة للـ Application. يجب ان يؤدى هذا التعيين فى وقت التشغيل.

- تقوم ملفات الـ Initialization بتخزين خيارات ومواصفات وقيم اخرى. استخدم الـ TIniFile class لإنشاء object يمكنك ان تستدعى له methods مثل الـ ReadSectionValues، والـReadBool، والـ WriteString لقراءة وكتابة مواصفات ملف الـ ini.

- توضح الـ SysColor utility الخاصة بهذا الباب الـ methods العملية للتعامل مع ملف الـ ini.. ويوضح البرنامج ايضاً تقنيات عديدة ومفاهيم برمجة Pascal.

يغطى الباب التالى Delphi dialog components، ويظهر افكاراً لكتابة الـ modeless dialog windows و modal.

*****

الباب الثانى عشر
التعامل من خلال
الـDialog Boxes

محتويات هذا الباب:

- Components
- Dialog modes
- Common dialogs
- Paged dialogs
- البحث من خلال الـ dialogs
- Page controls
- تحديد حجم النافذة
- انشاء docking controls

فى الـWindows، يعتبر الـ Dialog Box من الناحية الفنية نافذة ذات طرازاً محدداً تفتقد ازرار التكبير والتصغير، وغالبـاً من تحتوى على check boxes، radio buttons، وConrtols اخرى. ولكـن فى Delphi، أى form يمكن ان يكون لها سمات الـ dialog box، والخطوط الفاصلة بين dialogs التطبيق ونوافذه الأخرى تعتبر غير واضحة فى ضوء واجهة التطبيق الجرافيكية الحديثة للمستخدم.

وقد تعتقد ان الـ dialog box، كما اعنى هنا، هو اذن وصوت البرنامج. إنه يمنح المستخدمين وسيلة لإختيار الخيارات فى إدخال المدخلات. ويمكن للـdialog

box أن يرسل رسـائل، يسـأل أسـئلة، ويتلقى إجابات. فى هذا البـاب، سـوف تتـعـرف على الـ dialog components الخـاصـة بـ Delphi، وكـذلك الـ TabControl، الـ PageControl، والـ TabSheet، والتى جـعلت من السـهل إنشاء dialogs ونوافذ أخرى مع الـ page-tab controls فى مثل page tabs فى الـ Delphi VCL palette، و dialog boxes متعددة الصفحات. سوف أوضح أيضاً فى هذا الباب إثنين من تقنيات البرمجة الحديثة فى Delphi- كيفية إنشاء docking windows، وكيفية إضافة قيود على أحجام النوافذ.

وليكون مرجعاً كاملاً، يوضح هذا الباب أيضاً كيفية استخدام الـ Notebook، والـ TabbedNotebook، والـ TabSet الخاصة بالـ Win 3.1 القديم. يجب على التطبيقات الجديدة أن تستخدم الـ TabControl، والـ PageControl، والـ TabSheet ولكنك تحتاج معلومات عن هذه الـ components القديمة اذا كان تطبيقك يجب أن يتم تشغيله فى الـ Windows 3.1 أو اذا كنت تطور برمجيات قديمة لتستخدم components حديثة.

:Components

فيما يلى قائمة الـ dialog components العام الخاص بـ Delphi :

- **ColorDialog:** إن هذا الـ component يتضمن dialog اللون للـ Windows. استخدمه كأداة لإختيار اللون. Palette : Dialogs.
- **FindDialog:** يتضمن الـ search dialog للـ Windows. استخدمه لأداء بحث للبيانات. الـ Palette : Dialogs.
- **FontDialog:** يتضمن الـ font-selection dialog للـ Windows. استخدمه للسماح للمستخدمين بإختيار text Font، ولتحدد لون الواجهة للنص المعروض. Palette : Dialogs.
- **Notebook:** يقدم هذا الـ component حاوية متعددة الصفحات والتى يمكنك استخدامها مع الـ TabSet لتوفير toolbars مجدولة ومجموعة controls أخرى. Palette : Win3.1.

• **OpenDialog:** ويحتوى Open file dialog الخاص بالـ Windows . استخدمه لإختيار الملفات وتصفح الأدلة، كاستجابة نموذجية لأمر الـ File|Open . Palette : Dialogs.

• **PageControl:** يمكنك إنشاء page-tab بأقل برمجة مطلوبة . كل page فى الـ PageControl يعتبر ، TabSheet object منفصل . وعلى هذه الـ objects، يمكنك إسقاط controls أخرى مثل الأزرار، check boxes، radio-boxes- أى شئ تريده . يستطيع المستخدمون ضغط أبواب الـ PageControl page tabs للإختيار من بين صفحاتها والوصول للـ control الموجود على كل page . إن طريقة رائعة لوضع العديد من الـ controls على مسافة صغيرة- على سبيل المثال، فى برنامج ذى اختيارات كثيرة . Palette : Win32.

• **ReplaceDialog:** يتضمن الـ replace dialog الخاص بالـ Windows . إستخدمه لأداء عمليات البحث والابدال . Palette : Dialogs.

• **SaveDialog:** يتضمن الـ dialogSave-fil الخاص بالـ Windows . استخدمه لإدخال أو إختيار اسماء ملف، لتصفح الأدلة، كأستجابة لأمر الـFile|Save . Palette : Dialogs.

• **TabbedNotebook:** يقدم هذا الـ component الـ multipage panel-like object والذى لها ابواب ذات labels على الحافة العلوية يستطيع المستخدمون اختيار صفحات الـ TabbedNotebook بإختيار الـ labels، والتى تجعل هذا الـ component ملائم بشكل خاص لـ dialogs الخيارات المعقدة ذات الـcontrols العدد الكثير من بحيث ان تتناسب داخل نافذة واحدة . Palette : Win3.1.

• **TabControl:** تنشئ dialogs متعددة الصفحات ولكن، الـ TabControl يعتبر object واحد- ان "صفحاته" هى مجرد خداع، وهى المسئولة عن التحكم فيها فى وقت التشغيل . يمكنك استخدام الـ TabControl مع مجموعة اخرى من الـcontrols التى لها محتويات مختلفة- سلسلة من حقول الـ editing مثلاً- إعتماداً على الـ page-tab المختار . وهذا يختلف عن الـ PageControl والذى تكون فيه كل صفحة TabSheet منفصل يمكن ان يحمل مجموعة من الازرار و الـcontrols الاخرى . Palette : Win32.

● **TabSet:** يقدم الـ toolbar والذى يتكون من عدد من tabs ، والتى يمكنك استخدامها بالارتباط مع components اخرى لتزودهم بامكانية الإنتقال بين الـ pages . يوضح الـ TabEdit الخاص بهذا الباب كيفية استخدام الـ TabSet لإنشاء text editor متعدد الصفحات . Palette : Win3.1 .

● **TabSheet:** يقـوم الـ PageControl بتـخـزين واحـداً أو أكـثـر من الـ TabSheet objects ، والذى يحـمل الـ controls التى تظهـر على كل page من النافـذة أو الـ dialog متـعـدد الـ pages . لإنشـاء الـ TabSheet ، اضـغـط على الزر الأيمن للفـأرة داخل الـ PageControl واخـتـر أمـر الـ New Page من القـائمـة . Palette : None .

Dialog Modes :

ان التطبيقات تسـتـخدم dialogs بواحـدة من طريقـتين : كنوافذ الـ modal تحـتـفظ بالـ input focus حـتى الإغـلاق ، أو كنوافـذ modeless تسـمح للمستـخدمين بالتحول بعيداً عنها . على سبيل المثال ، يعتبر dialog الاختيارات نافذة modal يجب على المستخدمين إغلاقها قبل الاستمرار فى استخدام التطبيق . ولكن ، dialogs البحث والإبدال يعتبر modeless بحيث يستطيع المستخدمون بدء البحث ، التحول الى نافذة dialog ، ثم استكمال البحث .

Modal dialogs :

استدع الـ ShowModal method الخاص بالـ form لعرض الـ dialog . يجب على المستخدمين ان يغلقوا نافذة الـ dialog لاستكمال البرنامج . على سبيل المثال ، اذا كان الـ MyDialog يعد form ، فإن أمر قائمة أو OnClick لزر ما قد يعرض نافذة الـ dialog بالعبارة :

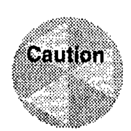
```
MyDialog.ShowModal;
```

فى المعتـاد ، تقوم ازرار الـ Close ، والـ Cancel ، أو ازرار إغلاق الـ form الاخرى الخـاصـة بالـ dialog بتـعـيين قـيـمـة للـ ModalResult ، الذى يقـوم الـ ShowModal بإدخاله . لتحديد ما اذا كان المستخدمين قد ضغطوا زر الـ OK للـ dialog ، استخدم عبارة مثل التالية لاختيار قيمة إدخال الـ ShowModal :

```
if MyDialog.ShowModal = mrOk then
{ ... take action if user clicked OK button }
```

* فى dialog modules، قم بتعيين القيمة التى تريد للShowModal ان يدخلها للـ ModalResult. إما ان تحدد mork لـ خاصية الـ ModalResult أو فى OnClick الخاص بزر الـ OK للـ OnClick استخدم عبارة مثل هذه :

```
ModalResult := mrOk;
```



تحذير: القيم التى تم تعيينها للـ ModalResult يجب ان تكون غير صفرية حتى يعود الـ ShowModal.

عرض انواع Objects اخرى من Modal Dialog:

يقوم الـ ShowModal بادخال قيمة عدد صحيح لإدخال من object انواع اخرى من modal dialog أضف هذه function إلى الـ form class لإدخال البيانات التى تحتاجها. على سبيل المثال، قم بتعريف String function فى القطاع الـ public للـ form class ،

```
TYourForm = class(TForm)
...
public
 function GetStringResult: String;
end;
```

نفذ الـ function لإرجاع نتيجة الـ dialog مثل خاصية الـ Text التابعة للـ Edit؛

```
function TYourForm.GetStringResult: String;
begin
 with Edit1 do
if Length(Text) = 0 then
  Result := 'Default string'
 else
  Result := Text;
end;
```

استدع الـ ShowModal لعرض نافذة dialog form ثم استدع function الـ "get result" مباشرة بعدها؛

```
if YourForm.ShowModal = mrOk then
 S := YourForm.GetStringResult;
```

: Modeless dialogs

كما يعرفه الـ Windows، الـ modeless dialog يعتبر Child Window لا تملك input focus وتسمح للمستخدمين بتشغيل التطبيق بشكل طبيعى اثناء عرض الـ dialog box على الشاشة. ولكن فى Delphi، لعرض form على انها modeless dialog، استدع الـ Show method :

```
MyDialog.Show;
```

ان استدعاء الـ Show يحدد خاصية الـ Visible للـ form بـ True. ويستدعى الـ BringToFront لعرض الـ dialog box فوق النوافذ الاخر- . لإخفاء نافذة الـ dialog، حدد الـ Visible بـ False :

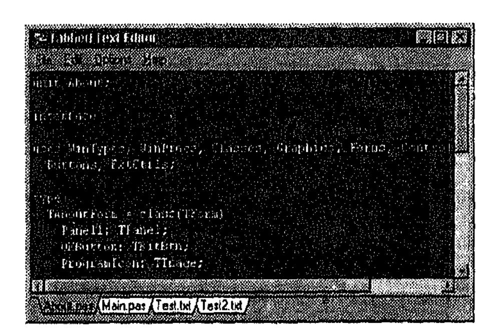
```
MyDialog.Visible := False;
```

لتحدد ما اذا كان modeless dialog نشط أم لا، افحص خاصية الـ Visible له.

: Common Dialogs

على القرص المدمج: ان اربعة من الـ common dialogs - وهى الـ FontDialog، ColorDialog، OpenDialog، والـــــ SaveDialog - توفر نوافذ اختيار، ولون، واسم ملف. يوضح تطبيق الـ TabEdit على القرص المدمج كيفية استخدام هذه الـ components، ان الـ source code للـ TabEdit طويلة جداً بحيث يصعب ذكرها هنا كاملة، لذا قدمت القوائم فى مقتطفات صغيرة نسبياً. بالطبع، توجد الـ source code كلها على القرص المدمج فى دليل الـ Source\TabEdit. يوضح الشكل (١٢-١) عـــرض الـ TabEdit. قم بتحميل ملف مشروع الـ TabEdit فى Delphi لفحص البرنامج كاملاً.

ملحوظة: يستخدم الـ TabEdit الـ Win3.1 components لإنشاء صفحاته المتعددة، التى يتم اختيارها بإمتياز الـ page tabs فى اسفل النافذة. مؤخراً فى هذا الباب، سوف اوضح كيفية استخدام الـ TabControl components، الـ TabSheet، والـ PageControl الحديثة لإنشاء dialogs ونوافذ متعددة الصفحات.



شكل (١٢-١): يقدم الـ TabEdit text editor متعدد الصفحات، مشابهاً للـ code editor الخاص بـ Delphi

: Font and Color dialogs

كما توضح القائمة (١٢-١)، ان components الـ ColorDialog والـ FontDialog سهلة الاستخدام. أضف هذه الـ objects على form استدع الـ Execute methods الخاصة بها لعرضها كـ modal dialogs. اذا عاد الـ Execute بـ True، فقد ضغط المستخدم الـ OK لإغلاق نافذة الـ dialog، ويمكنك نسخ أو استخدام خاصية الـ Font للـ FontDialog أو خاصية الـ Color للـ ColorDialog للحصول على اختيار المستخدم.

القائمة (١٢-١): ان components الـ ColorDialog والـ FontDialog يسهل استخدامها

```
procedure TMainForm.OptionsFontClick(Sender: TObject);
begin
  if FontDialog1.Execute then
     Memo1.Font := FontDialog1.Font;
end;

procedure TMainForm.OptionsBackgroundClick(Sender: TObject);
begin
  if ColorDialog1.Execute then
     Memo1.Color := ColorDialog1.Color;
end;
```



يوفر الـ ColorDialog ثلاث اختيارات، والتي يمكنك برمجتها باستخدام الـ Object Inspector أو بتعيين True أو False لثلاث خصائص اثناء التشغيل :

• **cdFullOpen**: عندما يكون هذا الاختيار محدد بـ True، يوفر الـ dialogs، grid لاختيار لون والتي يمكن للمستخدمين ان ينتقوا الالوان منها. عندما يكون الـcdfullOpen محدد بـTrue، يظهر dialog اللون لوحة الاختيار بها controls قيم اللون الاحمر والاخضر والازر وقيم الالوان الاخرى. على الحاسبات البطيئة نسبياً، قد تمر عدة ثوان قبل ان ينشيء الـ Windows الـ dialogs، فلا تستخدم هذا الاختيار اذا كنت تعلم ان تطبيقك يحتاج الى تشغيل على نظم بطيئة. مازال المستخدمون يفتحوا الـ dialog بضغط زر الالوان المخصص لذلك.

• **cdPreventFullOpen**: محدد ه بـ True لمنع المستخدمين من فتح dialog اختيار اللون (يكون زر الالوان المخصص دائماً بـ disabled). يمكنك استخدام هذا الخيار لقصر الالوان على مجموعة سابقة التحديد.

• **cdShowHelp**: حدد ه بـ True لعرض زر الـHelp. محدد ه بـ False اذا كان برنامجك لا ينفذ online help. اذا كان لدى برنامجك online help، حدد الـ cdShowHelp بـ True وعين قيمة مناسبة لخاصية HelpContext التابعة للـ dialog.

ويوفر الـ FontDialog خيارات عديدة، بعضها واضح- مثل الـ fdTrueTypeOnly والـ fdFixedPitchOnly. فيمايلى بعض الافكار عن الخيارات الأقل وضوحاً:

• **fdAnsiOnly**: حدد ه بـ True لعرض الـ font التى تنفذ مجموعة رموز الـ Windows فقط على سبيل المثال، عندما يكون الـ fdAnsiOnly محدد بـ True، لا يقوم font dialog بذكر الـ WingDings Symbol font.

• **fdEffects**: حدد ه بـ False لابطال خيارات اللون، underline، strikeout. يجب ان تقوم دائماً بايقاف عمل هذه العناصر اذا كان برنامجك لا يستخدمها. ولكن للاسف لا يمكن تشغيل وإبطال الخيارات الثلاثة اختيارياً (تحديد من الـ Windows). هذا الخيار للكل أو لا شئ.



• **fdNoFaceSel**: من الطبيعي، يعرض الـ font dialog الحالي في جزئية الـ edit للـ combo Box الخاص بقائمة الـ font. حدد الـ fdNoFaceSel بـ True اذا كنت تريد ان يكون edit control خالياً عندما يظهر الـ dialog.

• **fdNoSimulations**: حدده بـ True لمنع ذكر الـ font الشبيهه بالـ GDI في قائمة. وهذا المفتاح ليس له تأثير على الـ fonts الشاشة؛ فقط لـ fonts الطابعة.

• **fdNoSizeSel، fdNoStyleSel**: هذه المفاتيح تحدد ما اذا كانت اختيارات النمط والحجم قد تم اختيارها بداية (تم إبرازها). اذا كنت تريد ان يختار المستخدمون font جديد، ولكنك لا تريد عرض سمات الـ font الحالي، حدد هذان الاختياران والـ fdNoFaceSel بـ True.

• **fdWysiwyg** : حدده بـ True لذكر فقط الـ font المتاحة لكل من الطابعة والشاشة. عندما يكون الـ fdWysiwyg محدد بـ False، يستطيع المستخدمون اختيار fonts قد لا تطبع كما هي معروضة. اذا كان النص يظهر فقط على الشاشة ولا يطبع، حدد هذا الاختيار بـ False.

• **fdLimitSize** : حدده بـ True فقط عندما تحدد قيم خاصية الـ MinFontSize والـ MaxFontSize لقصر اختيارات الحجم على مدى محدد.

Note **ملحوظة**: يمكنك ايضاً استخدام الـ FontDialog لاختيار fonts مخصصة، أو متجانسة مع طابعة معينة.

Open and Save dialogs:

بوصفه text editor، يعتبر الـ TabEdit برنامجاً مفيداً في مجاله، ولكنه يعمل ايضاً كـ template لأي برنامج يتعامل مع ملف ان الخطوة الأولى هي كتابة procedures، واحداً لقراءة ملف من على القرص، والثاني لإنشاء أو كتابة بيانات ملف. إذن اكتب الـ procedures منفصلين دائماً، بدلاً من اخراج وادخال ملف الـ code في الـ event handlers لأمر القائمة هذا يجعل الـ procedure ملف البرنامج متاحة لأي نظم تحتاجها. توضح القائمة (١٢-٢) SaveFile والـ LoadFile للـ TabEdit.

ان الـ LoadFile و الـ SaveFile يستخدما الـPascal exceptions لتجربة عبارات قد تفشل. فى حالة الاخطاء، تقوم عبارة except بعرض رسالة الـdisplay. يوضح الباب التاسع عشر المزيد حول التعامل مع exceptions، ولكنه سهل الاستخدام. فقط ضع كل العبارات التى قد تولد exceptions- التابعة لـ procedure فى الـ except، وقم بإنهائه بكلمة الـ end الرئيسية. ان جزئية الـ on-do فى الـ except. تمسك بـ except objects معينة (الـ EReadError و الـ EWriteError فى هذه الحالة). من المهم ان تمسك فقط بالـ exceptions التى تريد التعامل معها، وتسمح للآخرين بالتعامل مع الـ exception handlers الافتراضية فى Delphi.

القائمة (١٢-٢) Tab Edit's file input and output procedures:

```
{ Read file from disk }
procedure TMainForm.LoadFile(const Path: String);
begin
 with Pages[TabSet1.TabIndex] do
    try
      Memo1.Lines.LoadFromFile(Path);
         Dirty := False;
         Page.Clear;
         SetFilename(Path);
        except on e: EReadError do
      MessageDlg('Error reading file', mtError, [mbOk], 0);
        end;
end;

{ Write current file to disk }
procedure TMainForm.SaveFile(Index: Integer);
begin
 with TabSet1, Pages[Index] do
        begin
         try
             Memo1.Lines.SaveToFile(Filename);
                Dirty := False;
              except on e:EWriteError do
```

```
          MessageDlg('Error writing file', mtError, [mbOk], 0);
            end;
        end;
    end;
```

توضح القائمة (١٢-٢) الـ event handlers لقائمة File للـ TabEdit. تستخدم الـ procedures الـ SaveDialog components والـ OpenDialog، وتستدعى الـ procedures الموجودة فى القائمة (١٢-٢). استخدم هذه القائمة كمرشد لك فى إضافة أوامر الـ Open والـ Save والـ Save As لتطبيقاتك.

القائمة (١٢-٢) : الـ event handlers لقائمة File للـ TabEdit

```
{ File|Open command }
procedure TMainForm.FileOpenClick(Sender: TObject);
begin
 with Pages[TabSet1.TabIndex] do
    begin
      if Dirty then FileSaveClick(Sender);
        if {still} Dirty then Exit;  { File not saved }
        if FileOpenDialog.Execute then
          LoadFile(FileOpenDialog.Filename);
      end;
end;

{ File|Close command }
procedure TMainForm.FileCloseClick(Sender: TObject);
var
    W: Word;
begin
 with TabSet1, Pages[TabIndex] do
    begin
      if Dirty then
         begin
          W := MessageDlg(
              'Save changes to ' + Tabs[TabIndex] + '?',
              mtWarning, [mbYes, mbNo, mbCancel], 0);
```

```
        case W of
                mrYes: FileSaveClick(Sender);
                mrNo: Dirty := False;
                mrCancel: Exit;
        end;
            end;
        if {still} Dirty then Exit;  { File not saved }
        Page.Clear;
        Memo1.Clear;
        Filename := untitledName;
        Tabs[TabIndex] := Filename;
    end;
end;

{ File|Save command }
procedure TMainForm.FileSaveClick(Sender: TObject);
begin
 with TabSet1, Pages[TabIndex] do
    if Filename = untitledName then
     FileSaveAsClick(Sender)
       else
     SaveFile(TabIndex);
end;

{ File|Save As command }
procedure TMainForm.FileSaveAsClick(Sender: TObject);
begin
 with TabSet1, Pages[TabIndex] do
    if FileSaveDialog.Execute then
    begin
     SetFilename(FileSaveDialog.Filename);
        SaveFile(TabIndex);
       end;
end;
```

ان الـ FileOpenClick يحـفظ الملف الحالى اذا كـان الـ Dirty محـدد بـ True، مما يشير الى ان المستخدم قد قام بتغييرات للملف. بعد هذا العمل، يعرض الـprocedure الـ FileOpenDialog باستدعاء الـ Execute method الخاص به. اذا حدد الـmethod بـ True يقوم الـ LoadFile عندئذ بقراءة الـ Filename المختار التابع للـdialog.

يحث الـ FileCloseClick المستخدمين على حفظ اية تغييرات وإغلاق الملف الحالى. ان الـ FileSaveClick يتحول الى الـ FileSaveAsClick اذا كان الملف الحالى لا يحمل اسماً: وإلا يستدعىprocedure الـ SaveFile. لمحاولة كتابة الملف الحالى على قرص. لا يستعمل أى من هذين الـ procedure أى dialog components.

يقوم الـ FileSaveAsClick باستدعاء الـ Execute لـ FileSaveDialog object. اذا حدد الـ method بـ True، يكون المستخدمون قد اختاروا اسم ملف جديد (أو سمحوا بالكتابة على ملف موجود)، ويستدعى البرنامج الـ SaveFile لكتابة الملف الحالى على قرص. إن عبارة الـ SetFilename تعين اسم الملف لـ pagetab label.

أنواع الملفات المتعددة و Filters:

فى برنامج يفتح أنواع مختلفة من الملفات، من الأفضل دائماً أن تقتصر على الـ SaveDialog filters على نفس نوع الملفات الأحدث الذى تم فتحه. على سبيلُ المثال، اذا فتح المستخدم ملف نص، يمكن أن يعرض الـ SaveDialog ملفات الـ txt. فقط. اذا فتح المستخدم bitmap، سوف يعرض الـ SaveDialog ملفات الـ bmp. هذا يذكر المستخدم بنوع الملفات التى يفتحها ويحفظها. اذا قمت بدلاً من ذلك ببرمجة نفس الـ filters فى كلاً من الـ OpenDialog والـ SaveDialog، قد يتخيل المستخدم أنه بإمكانه التحول بين أنواع الملفات ببساطة بواسطة تغيير الإمتداد (على سبيل المثال، بحفظ ملف الـ txt. على أنه ملف bmp.). اذا لم يكن بإستطاعة تطبيقك أن يؤدى مثل هذه filters، فمن الأفضل أن تعتقد أنه يستطيع.

. لإعادة برمجة الـ SaveDialog Filter، استخدم العبارة التالية بعد تنفيذ الـ OpenDialog object، عندما يحاول المستخدم فى المرة التالية أن يحفظ الملف،

يعرض الـ SaveDialog فقط الملفات التى لها نفس الإمتداد . على سبيل المثال، اذا فـتـحت ملف bmp. ، سـيكون الـ string الناتج المعين لخاصيـة الـ Filter لـ SaveDialog سيكون Files *.bmp|(*.bmp) :

```
SaveDialog1.Filter :=
 'Files (*'
   + ExtractFileExt(OpenDialog1.FileName)
   + ')|*'
   + ExtractFileExt(OpenDialog1.FileName);
```

فكرة: باستخدام الـ Format function الـ arguments المكررة فى الأقواس المربعة، يمكن تقصير العبارة السابقة بحذف الإستدعاء المزدوج للـ ExtractFileExt . على سبيل المثال، هذه العبارة مساوية للـ code السابقة :

```
SaveDialog1.Filter := Format(
 'Files (*.%s)|*.%0:s',
   [ExtractFileExt(OpenDialog1.Filename)]);
```

إنشاء قائمة تاريخ:

ان الـ OpenDialog والـ SaveDialog لهـا خـاصـيـة TStrings ، وهى الـ HistoryList ، والتى يمكنك استخدامها لحفظ اسماء الملف التى تم اختيارها حديثاً. هذا ينشئ تاريخاً من اسماء الملفات والتى يمكن للمستخدم ان يختار منها الملفات التى عمل بها سابقاً. يمكنك حتى حفظ قائمة التاريخ فى ملف لإستعادته فى المرة التالية لبدء البرنامج .

ان إنشاء قائمة تاريخ يتطلب ثلاث خطوات :

١- أضف الـ SaveDialog أو الـ OpenDialog على from .

٢- قم بتغيير خاصية الـ FileEditStyle للـ SaveDialog أو الـ OpenDialog لتصبح fsComboBox .

٣- أضف اسماء الملفات المختارة لخاصية الـ HistoryList للـ SaveDialog أو الـ OpenDialog .

لإتمام الخطوة رقم ٣ ، فى OnClick أو button event hundler والتى تفتح أو نضغط الملف ، قم بتنفيذ الـ dialog واضف اسم ملف مختار لقائمة التاريخ بـ code مثل هذه :

```
with OpenDialog1 do
if Execute then
begin
  { Open file here }
    FileName := Lowercase(FileName);
    HistoryList.Add(FileName);
end;
```

يمكنك فعل نفس الشئ بالـ SaveDialog . عندما يقوم المستخدم بفتح الـ dialog فى المرة التالية ، تظهر الملفات الاحدث اختياراً فى نافذة الـcombobox edit . (جرب الـ code السابقة فى OnClick الخاص بالزر لترى التأثير تأكد من تحديد خاصية الـ FileEditStyle للـ dialog بـ fsComboBox ، أولاً تظهر قائمة التاريخ) .

لتحديد عدد اسماء الملفات فى الـHistoryList ، قم بتعريف ثابت فى الـ module:

```
const
  maxHistoryList = 6;
```

ثم ، استخدم البرمجة التالية لإدخال اسماء ملف فى الـ HistoryList . تظهر الملفات الاحدث اختياراً على رأس القائمة ؛ والملفات القديمة تتحرك الى اسفل عندما تصل القائمة الى اقصى حجم لها :

```
with OpenDialog1 do
if Execute then
begin
  { Open file here }
    FileName := Lowercase(FileName);
    with HistoryList do
    begin
      if Count = maxHistoryList then
        Delete(Count - 1);
          HistoryList.Insert(0, FileName);
      end;
end;
```

قد تريد ايضاً ان تتضمن امراً أو اختياراً لمسح قائمة التاريخ . لفعل هذا ، استخدم عبارات مثل هذه :

```
OpenDialog1.HistoryList.Clear;
SaveDialog1.HistoryList.Clear;
```

: Paged Dialogs

بالرغم من ان الشاشات اليوم ذات تقنية عالية ، الا ان المساحة المتوفرة على شاشة تعتبر قليلة ونادرة . لتجد بعض المساحات الإضافية لـ controls و objects اخرى ، يمكنك إنشاء dialog boxes متعددة الصفحات . هذا يساعد ايضاً على تنظيم الـcontrols في فئات يختارها المستخدمون باختيار tabs ذات labeled . تقترح الفصول التالية سبل إنشاء نوافذ بـ components الـ TabSet ، الـ Notebook ، والـ TabbedNotebook .

Note

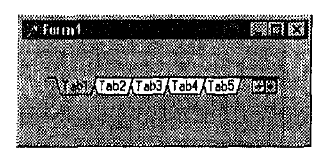
ملحوظة: الـ components الموضحة في هذا الفصل توجد على الـ Win3.1 palette ، وهي متاحة لبرامج الـ Windows ذات الـ ٣٢ والـ ١٦ بت . البرامج الاحدث يجب ان تستخدم الـ PageControl والـ TabShee والـTabControl الموضحة بعد ذلك في هذا الباب . ولكن المعلومات المذكورة هنا عن الـ Win3.1 components مازالت ذات قيمة اذا كنت تحتاج ان تدعم الـ Win3.1 ، أو اذا كنت تقوم بتطوير تطبيقاً جديداً لتستخدم الـ controls الحديثة .

الـTabSet:

يوضح شكل (١٢-٢) الـ TabSet object في نافذة اختيارية لإنشاء الـ object ، قمت بضغط الزر البيضاوى لخاصية الـTabs ، وادخل سبعة labels ، Tab1 ، Tab2 ، . . . ، Tab7 . ولكن ، بسبب انه لايوجد الا اماكن لستة labels من السبع ، يعرض الـ component ازرار اسهم افقية يمكن للمستخدم ضغطها لتحريك الـ TabSet يميناً ويساراً .

يستخدم تطبيق الـTabEdit الـ TabSet object ، محاذياً للحافة السفلية النافذة ، لإنشاء الـ Memo . المتعدد الصفحات . في الواقع ، ان للبرنامج Memo

object واحد . لتخزين ملفات متعددة في ذاكرة ، يستخدم البرنامج الـ Pages array العام لسجلات الـ TPageRec . يحتوى كل سجل TPageRec على TStringList object (Page) . عندما تختار tab لتقلب الصفحة ، يعين البرنامج قائمة الـ string المختارة لـMemo object .



شكل (١٢-٢): يعرض الـ TabSet ازرار اسهم افقية بحيث يمكن للمستخدمين تحريك الـTabs يساراً ويميناً اذا لم يكن هناك مساحة كافية لكل Tabs

توضح القائمة (١٢-٤) TabSet event handlers التابعة للبرنامج . فى اغلب الحالات ، إنك تحتاج الى ان تستجيب لـ events فقط - واحداً تم توليده قبل تغير الـ Tab وواحداً تم توليده بعد ذلك . ان الـ two events يعطيان برنامجك الفرصة لفصل الـ page الحالية عندما يكون الـ Tab على وشك ان يتغير ثم يلحق صفحة جديدة بعد التغيير . ان معنى الصفحة يعتمد كلية على تطبيقك - ان الـ TabSet يعطيك وسائل منضبطة لإنشاء الخدع متعدد الصفحات .

القائمة (١٢-٤) : الـ event handlers التابعان للـ TabSet فى تطبيق الـ TabEdit

```
{ A tab is changing. Save Memo's text in a TStringList object }
procedure TMainForm.TabSet1Change(Sender: TObject; NewTab:
Integer;
  var AllowChange: Boolean);
begin
  with TabSet1, Pages[TabIndex] do
    begin
      Page.Clear;
        Page.Assign(Memo1.Lines);
      end;
end;

{ A tab has changed. Assign a TStringList object to Memo }
procedure TMainForm.TabSet1Click(Sender: TObject);
```



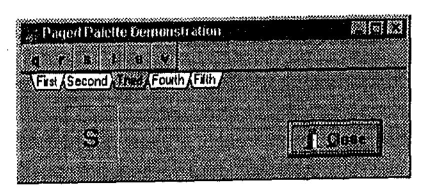
```
begin
  with TabSet1 do
      Memo1.Lines.Assign(Pages[TabIndex].Page);
end;
```

يقـــوم الـ TabSet1Change Procedure اولاً بمسح الـ TStringList Page الحالى . بعد ذلك، يستدعى الـ Procedure الـ assign لكل Page، مروراً بخـاصـيـة الـ Lines للـ Memo (object من نوع الـ TStrings) . هذا يحـفظ المحتويات الحالية للـ Memo فى الـ Page قبل ان يتغير الـTabSet .

بعد التغيير، يقوم الـ TabSet1Click method مرة اخرى باستدعاء الـ Assign . ولكن هذه المرة، يذهب التـعـيين فى الاتجـاه العكسى- من Page TStringList المفـهرسة بالـTabIndex لقائمة الـ Lines TStrings التابعة للـ Memo. الآن، يعرض الـ Memo الصفحة الجديدة من النص المرتبط بالـ tab المختار.

الـNotebook :

على القرص المدمج: يقدم الـ Notebook حاوية متعددة الصفحات والتى تستخدم فى اغلب الاوقات مرتبطة بالـ TabSet، وللحصول على مثال عملى حول هذه التقنية، جرب تطبيق الـ Palette على القرص المدمج فى دليل الـ Source\Palette . مع اثنين من الـevent handlers فقط، ينشئ البرنامج toolbar، يشبه الـ VCL palette الخاصة بـ Delphi يوضح شكل (١٢-٣) مظهر الـpalette . اضغط page tab للانتقال الى صفحة اخرى من الـSpeedButtons . اختر أى SpeedButtons لعرض الـ caption الخاص به فى الـ Label. توضح القائمة (١٢-٥) الـ source code للبرنامج .



شكل(١٢-٣)يوضح تطبيق الـpalette كيفية إنشاءpage tab، يشبه الـ VCL palette التابعة بـ Delphi

لاختصار المساحة فى هذا الباب، قمت بحذف اغلب تعريفات الـSpeedButtons . الـ ٣٦ من الـ TMainForm class .

القائمة (١٢-٥): Palette\Main.pas

```
unit Main;

interface

uses
  SysUtils, Windows, Messages, Classes, Graphics,
    Controls, Forms, Dialogs, Buttons, ExtCtrls, Tabs,
  StdCtrls;

type
  TMainForm = class(TForm)
      TabSet1: TTabSet;
        Notebook1: TNotebook;
        SpeedButton1: TSpeedButton;
        SpeedButton2: TSpeedButton;
    { ... }
      SpeedButton36: TSpeedButton;
        BitBtn1: TBitBtn;
        Label1: TLabel;
        Bevel1: TBevel;
        procedure TabSet1Click(Sender: TObject);
        procedure SpeedButton1Click(Sender: TObject);
      private
      { Private declarations }
      public
      { Public declarations }
      end;

var
    MainForm: TMainForm;

implementation
```

```
{$R *.DFM}

procedure TMainForm.TabSet1Click(Sender: TObject);
begin
  with TabSet1 do
    NoteBook1.ActivePage := Tabs[TabIndex];
end;

procedure TMainForm.SpeedButton1Click(Sender: TObject);
begin
  with Sender as TSpeedButton do
     Label1.Caption := Caption;
end;

end.
```

اتبع هذه الخطوات لإعـادة إنشـاء برنامج الـpalette وتكوين الـ tabled toolbar باستخدام الـNotebook :

١-أضف الـ Notebook على الـ form، وحـدد خـاصـيـة الـ Align به - alTop . قم بتغيير الـHeight الخاص بالـNotebook الى ٢٥ .

٢- أضف TabSet على الـ form . حدد خاصية الـ Align له بـ alTop . هذا يضع الـTabSet تحت الـNotebook مباشرة .

٣- اختر خاصية الـ Tabset's Tabs قم بتغيير الـ Tabs للـ TabSet1، واضغط الزر البيضاوى لتشغيل الـString list editor الخاص بـ Delphi. ادخل toolbar labels، واحـدة لكل سطر (لإعـادة إنشـاء البـرنامج، إدخل First، Second، Third، Fourth ، وFifth).

٤- اختر خاصية الـPages للـ Notebook1، واضغط الزر البيضاوى لفتح الـ Notebook editor الخـاص بـ Delphi . اضـغط زر الـEdit لتـعـديل اسم الصفحة الافتراضى ليصبح First . ثم، اضغط زر الـ Add لإضافة صفحات الـ tab labels المتبقية . عند ربط الـNotebook بـTabSet باستخدام التقنية الموضحة هنا، بطاقة لكل Notebook page يجب ان يكون TabSet مناسبة .

٥- ان خاصية الـ ActivePage للـ Notebook تساوى اسم الصفحات التى ادخلتها فى الخطوة السابقة . قم بتغيير الـ ActivePage لـ First . ادخل SpeedButton objects فى الـ Notebook1 (اضغط الفأرة داخل الـ Notebook1، ثم اسحب كل SpeedButton لموضعه النهائى).

٦- اعد الخطوة رقم ٥ لصفحات Notebook اخرى. قم بتغيير الـ ActivePage لـ Second، Third ، وFifth . . . . ، وإدخل بعض الـ SpeedButton فى كل صفحة . يمكنك وضع label على الـ SpeedButton الخاص بك كما فعلت فى المثال، أو قم بتعيين glyph bitmaps لهم. ادخل Label و اختر حجم font كبير لإظهار الـcaption الخاص بالزر المختار.

٧- عندما تنتهى من اعداد كل صفحة Notebook ، حدد الـ ActivePage بـ First وخاصية الـ PageIndex للـ TabSet بصفر. قم بإنشاء OnClick لكل الـ SpeedButton objects. ايضاً قم بإنشاء OnClick للـ TabSet1 . إدخل العبارات الموجودة فى القائمة. لتقلب الـ Page على الـ toolbar يقوم الـ TabSet1 handler يتعين الـ label الـ tabled للـ ActivePage الخاصة بالـ Notebook1. يقوم للـ SpeedButton handler بعرض الـcaption الخاص بالزر المختار.

Tip **فكرة:** لإنشاء tabled toolbar فى اسفل نافذة، اعكس الخطوات رقم ١ و ٢ وحدد الـ Align لكلا الـ objects بـ alBottom.

الـTabbedNotebook:

ان الاختلاف الاساسى بين الـTabbedNotebook وزوج من الـNotebook والـTabSet هو المظهر. استخدم الـTabbedNotebook لإنشاء dialogs متعددة الصفحات تبدو مثل file folders ذات tabs اختيار على طول القمة. استخدم الـNotebook والـTabSet عندما تحتاج مزيداً من التحكم على موضع الـ object، وعندما تحتاج مرونة برمجة عملية الإنتقال بين الـ Page باستخدام components منفصلة.

يعتبر الـTabbedNotebook مفيداً بشكل خاص فى إنشاء dialogs اختيارات- على سبيل المثال، مع طباعة الخيارات على صفحة واحدة، Color

Setup على اخرى، وخيارات أخرى على صفحة اخرى. بتقسيم الـ controls الى فئات، يمكنك برمجة امر الـ Options الذى يفتح نافذة واحدة، والتى قد يجدها المستخدم اقل تداخلاً من الأوامر المتعددة فى قائمة Options .

لاستخدام الـTabbedNotebook أضفها على الـ form . عادة ما تريد ان تحدد خاصية الـ Align للـ object بـ alClient حتى يملأ الـTabbedNotebook تماماً النافذة. ولكن هذا يرُجع إليك قطعاً. اختر خاصية الـ Pages لانشاء كل صفحة ذات labeled باستخدام نفس الـ Notebook editor المذكور فى الفصل السابق. على كل صفحة، إدخل الـ controls التى تحتاجها (CheckBoxes، RadioButtons، و objects اخرى).

فى وقت التشغيل، يمكنك عرض صفحة معينة بطريقتين، غالباً لن تحتاج ان تفعل هذا الآن فالمستخدمون يمكن ان يختاورا الصفحات التى يريدونها. ولكن، لعرض صفحة بالرقم، قم بتعيين قيمة للـ PageIndex. ان الفهرس الأول هو صفر، لذا فيمثل رقم اثنين الصفحة الثالثة :

```
TabbedNotebook1.PageIndex := 2;  {Display page 3}
```

أو، لعرض صفحة باسم الـ labeled، قم بتعيين الـstring للـ ActivePag :

```
TabbedNotebook1.ActivePage := 'General options';
```

للحصول على نافذة أكثر ترتيباً فى dialog الخيارات المعقد ذى الصفحات الكثيرة، قم بتعيين قيمة للـ TabsPerRow . على سبيل المثال، اذا كان الـ dialog الخاص بك له ١٢ صفحة، حدد الـ TabsPerRow بـ ٤ لتجميع الـ folders فى ثلاث صفوف من اربع. ولكن هذا يبدو انه يعمل فقط فى Delphi Versions 1 and 2. تعرض Versions 3 and 4 صفاً واحداً فقط من الابواب الـ tabs (انظر الـ PageControl فى هذا الباب لمعرفة طريقة افضل لإنشاء dialog متعددة الصفحات). وهذا لان Delphi 3 قد ترك code الـ ١٦ بت القديمة للـ TTabbedNotebook، واستبدلها بـ Tabs control ذى الـ ٣٢ بت من الـ Windows. وهذا التغيير قد أدى الى تحديد مظهر الـ control، وكذلك قلل حجم ملف الـ code. ولكن كنتيجة لذلك، لم يعد الـ component يستطيع تحديد عدد

الـ tabs لكل صف ولم يعد ممكناً تشغيل وابطال tabs منفردة. مازال الـ control قادراً على صفوف tabs متعددة- لا يمكنك فقط تحديد كم تريد. وهذه الجوانب السلبية تعتبر ضئيلة، والطفرة فى الأداء وقدرة الذاكرة على الحفظ أكبر بكثير من أى عيوب تنتج عن هذه. التغييرات.

Tip

فكرة: يقوم Delphi بصورة تلقائية بتعديل الـ TabsPerRow بحيث تتناسب كل الـ labels مع الـ tabs . لهذا السبب، يجب ان تعين قيمة جديدة لإنشاء كل صفحات الـ TabbedNotebook .

لتغيير font النظام الافتراضى لـ tab labels، اختر font مختلفاً لخاصية الـ TabFont . تؤثر خاصية الـ Font للـ control objects المضافه لـ TabbedNotebook . قد يكون الـ fonts مختلفين. على سبيل المثال، يمكنك استخدام نص مائل لـ tab labels ونص عادى للـ control على الـ tabbed Pages .

قد تنتمى الـ component objects الى form حتى عندما تبدو كأنها مستقرة داخل الـ TabbedNotebook . على سبيل المثال، أضف زر Close على الـ form واسحبه فوق الـ TabbedNotebook . يظهر الزر فى أعلى صفحة الكتيب، ولكنه مازال ينتمى للـ form . وبسبب علاقته بالـ Parent object، يبقى الزر مرئياً فى كل صفحات الكتيب. وهذا يعمل حتى عندما يملأ الـ TabbedNotebook تماماً الـ client area بالـ form .

بعد تحديد خاصية الـ Align للـ TabbedNotebook بـ alClient، من الصعب ان تضيف objects على الـ form، ولكن هناك حل سهل. حدد مؤقتاً الـ Align بـ alNone . اختر الـ Button أو الـ BitBtn (أو أى component آخر تريد إضافته) فى الـ VCL palette ثم قم بتكبير نافذة الـ form . اضغط مؤشر الفأرة فى خلفية الـ form لإضافه الـ object الخاص بك، واسحب الزر الناتج على أى صفحة TabbedNotebook (لايهم أى واحدة). قم بتقليص النافذة مرة اخرى لحجمها الطبيعى، وحدد خاصية الـ Align للـ TabbedNotebook مرة اخر بـ alClient . عندما تقوم بتشغيل البرنامج، يبقى الزر مرئياً بغض النظر عن أى صفحة من الصفحات تم اختيارها.

أو، استـمـر فى ضـغط مـفـتـاح Shift عندمـا تضـغط الـ form لاسـقـاط الـ component . ان ضـــغـط الـ Shift يســـقـط الـ object فى الـ Parent object للحاوية المستهدفة (الـ form فى الغالب) بدلاً من الحاوية نفسها .

ان احدى الاسـتـخـدامـات العـامـة للـ TabbedNotebook هى حـمـل الـ Memo، واحــداً فى كل صـفـحـة . إدخل الـ Memos فى صـفـحـات الـ TabbedNotebook، وحدد خصائص الـ Align للـ Memo بـ alClient . ولأن الـ TabbedNotebook، يملك الـ Memos، فهى تملأ الصفحة ولكن لا تؤثر على الـ tabbed labels . وهذه طريقة سريعة لإنشاء text editor متعدد الصفحات، ولكن عليك ان تقوم ببرمجة الـ method بحذر لأداء عمليات على الصفحة الحالية فقط . على سبيل المثال، لنسخ نص الى الـ clipboard، افحص الـ PageIndex لتحديد أى Memo تستخدم :

```
case TabbedNotebook1.PageIndex of
  0: Memo1.CopyToClipboard;
     1: Memo2.CopyToClipboard;
     2: Memo3.CopyToClipboard;
end;
```

هذا الحل يعمل ولكنه ليس الأمثل . توضح القـائمـة (١٢-٦) طريقـة أكـثـر عمومية تعمل بغض النظر عن عدد الصفحات التى يملكها الـ TabbedNotebook . يقـوم البـرنامج بتـعـيين المتـغـيـر P، من نوع الـ TComponent، نتـيـجـة الـ FindComponent، التى تبحث عن الـ Memo باستخدام الـ PageIndex لإنشاء اسمـاء الـ object (Memo1، Memo2، . . . ، MemoN) . فـإذا نجح البـحـث، يقوم البرنامج باستدعاء الـ CopyToClipboard للـ Memo الموجودة على الصفحة الحالية . قد تستخدم code شبيهة فى أمر قائمة الـ Edit|Copy أو OnClick للزر .

القائمة (١٢-٦): لنسخ نص Memo مختار الى الـ Clipboard، استخدم الـ FindComponent لوضع الـ Memo على صفحة الـ TabbedNotebook الحالية، ثم استدع الـ CopyToClipboard للـ Memo

```
procedure TForm1.Copytoclipboard1Click(Sender: TObject);
var
```

```
    P: TComponent;
  begin
   P := FindComponent('Memo' +
       IntToStr(TabbedNotebook1.PageIndex + 1));
         if P <> nil then
       TMemo(P).CopyToClipboard;
  end;
```



فكرة: ان اختيار component بقائمة اللائحة الخاصة بالـ Object Inspector ينتقل بصورة تلقائية الى صفحة الـ Notebook أو الـ TabbedNotebook التى تحتوى هذا الـ object .

إدخال صفحات اثناء وقت التشغيل:

على القرص المدمج: ان تطبيق الـ AddPage على القرص المدمج فى دليل الـ Source\AddPage يوضح كيفية إضافة صفحات جديدة الـ TabbedNotebook فى وقت التشغيل . يوضح البرنامج ايضاً كيفية إدخال contains control فى صفحة- مرة أخرى ، تحت تحكم البرنامج تماماً. تظهر القائمة (١٢-٧) الـ source code البرنامج . قم بتشغيل الـ AddPage واضغط الـ AddPage لإدخال صفحات جديدة فى الـ TabbedNotebook . اضغط الـ Add control لإدخال ListBox فى الصفحة الحالية .

القائمة (١٢-٧): AddPage\main.Pas

```
  unit Main;

  interface

  uses
   SysUtils, Windows,  Messages, Classes, Graphics,
      Controls, Forms, Dialogs, Buttons, StdCtrls, TabNotBk;

  type
      TMainForm = class(TForm)
```

```
      TabbedNotebook1: TTabbedNotebook;
        AddPageButton: TButton;
        CloseBitBtn: TBitBtn;
        AddControlButton: TButton;
        procedure AddPageButtonClick(Sender: TObject);
        procedure AddControlButtonClick(Sender: TObject);
    private
      { Private declarations }
       public
      { Public declarations }
       end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}

{ Insert new page into TabbedNotebook }
procedure TMainForm.AddPageButtonClick(Sender: TObject);
begin
  with TabbedNotebook1 do
      Pages.Add(Format('Page %d',[Pages.Count-1]));
end;

{ Insert new control into current page }
procedure TMainForm.AddControlButtonClick(Sender:
TObject);
var
  L: TListBox;
    P: TWinControl;
begin
  L := TListBox.Create(Self);
    with TabbedNotebook1 do
    begin
```

```
        P := Pages.Objects[PageIndex] as TWinControl;
          L.Parent := P;
          L.SetBounds(10, 10, 100, 100);
    { Insert page tab label into edit control for demonstration.
      You don't have to perform this step. }
        L.Items.Add(TTabPage(P).Caption);
          end;
    end;

    end.
```

يوضح الـ AddPageButtonClick Procedure كيفية إضافة صفحة للـ TabbedNotebook . ببساطة، اضف String لخاصية الـ Pages. يستدعى البرنامج الـ Format لإنشاء string labels مثل Page 1، Page 2 ، . . . ، Page n.

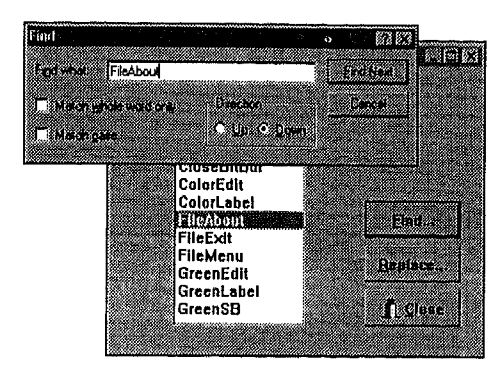
يوضح الـ AddPageButtonClick Procedure كيفية إضافة control object جديد لصفحة الـ TabbedNotebook الحالية. أولاً، ينشئ الـ Create constructsالـ الجديد. ثم بعد ذلك، يتم تحديد الـ P لتشير الى الصفحة، باستخدام الـ Objects array فى قائمة string التابعة للـ TabbedNotebook Pages. يعين البرنامج الـ Page reference (P) كـ control's parent، ثم يستدعى الـ SetBounds لتحديد مكان، وارتفاع، وعرض الـ control. (والانواع الأخرى من الـ control objects تتطلب انواع بدايات اخرى). اخيراً، لتعرض شيئاً فى الـ control الجديد، يضاف الـ Add method لـ tab label الصفحة الحالية للـ ListBox الذى تم إدخاله حديثاً. لا يجب عليك ان تؤدى هذه الخطوة الأخيرة.

من خلال الـ Dialogs:

ان الـ ReplaceDialog والـ FindDialog التابعة لـ Delphi سهلة للغاية. الجزء الصعب نوعاً ما هو كتابة code البحث والابدال التى تعمل بالارتباط بـ modeless dialog boxes. بعد هذه المقدمات القصيرة، يقدم تطبيق غلافاً يمكنك استخدامه لتنفيذ اوامر الـ Replace والـ Find.

الـFind dialog:

يقـــــدم الـ FindDialog component أثنان من methods و event. واحـد. لعـرض نافـذة الـ display الموضـحـة فى الشكل (١٢-٤) وبدء البـحـث، استدع الـ Execute method : استدع الـ CloseDialog لاخفاء الـ display. لاداء البحث الفعلى، اكتب للـ OnFind، الذى تم توليده عندما ضغط المستخدمون زر الـ Find Next.



شكل (١٢-٤): الـ FindDialog component يعرض modeless dialog.

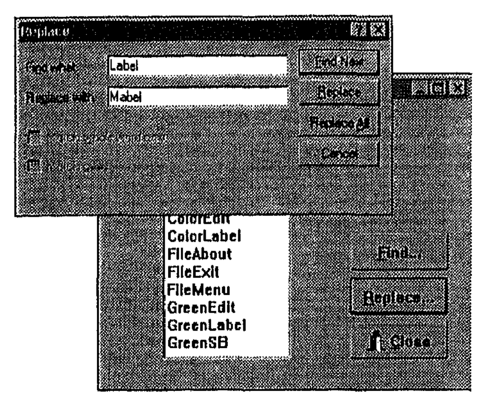
للحـصـول على النص الذى ادخله المسـتـخـدمـون فى الـ Find لأى edit control، اسـتـخـدم خـاصـيـة الـ Find Next للـ FindDialog. يمكنك كـذلك اختيار Options متنوعة- على سبيل المثال، لإخفاء وابطال خيار الـ match case check Box.

find and replace dialog:

كمـا هو موضح فى شكل (١٢-٥)، يعـد الـ ReplaceDialog نسـخة ممتدة للـ FindDialog، بالإضـافـة إلى edit control ثانى، لادخـال نص الابدال. لاعـضـاء الـFindDialog، يضـيـف الـ ReplaceDialog خـاصـيـة اخـرى، وهى ReplaceText، والتى تحـتـوى على النص الذى ادخله المسـتـخـدمـون فى الـ Replace بنافـذة الـedit. اسـتـدع الـ Execute لعـرض الـ ReplaceDialog. استدع الـ CloseDialog لإخفاء نافذة الـ display.

يتـعـرف الـ ReplaceDialog على events. إنشىء OnFind event handler للبـحث عن بنود النص. إنشىء OnReplace event handler لأداء

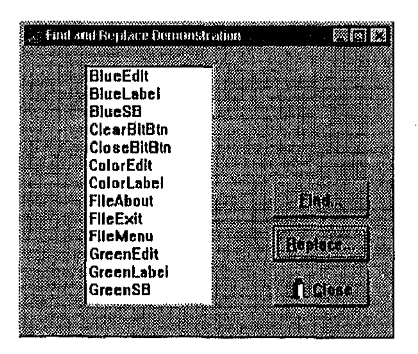
الإبدال ان كتابة هذا الـ code يتطلب ايضاً بعض الحيل، كما سأوضح فى الفصل التالى .



شكل (١٢-٥): الـ ReplaceDialog component يعرض.. modeless dialog.

برمجة اوامر Find والـ Replace :

على القــرص المدمج: لســوء الحظ ان برمــجــة اوامــر الـ Find والـ Replace اصعب بكثير من مجرد إضافة إثنين من الـ components على form . لتبسيط الأمر، اتبع الخطة الموضحة فى تطبيق الـ FindRepl على القرص المدمج . يوضح شكل (١٢-٦) البرنامج . إنه يذكر بعض اسماء الـ object (تم قصها من الـ SysColor Utility) فى الـ ListBox. اضغط زر الـ Find لتجد العناصر فى الـ ListBox اضغط Replace لتستبدل البنود أو جزء منها . توضح القائمة (١٢-٨) كيف يقوم البرنامج باداء هذه العمليات .



شكل (١٢-٦): يوضح تطبيق الـ FindRepl البرمجة اللازمة لأوامر الـ Find والـ Replace التابعة للتطبيق

القائمة (١٢-٨): FindRepl\Main.pas

```
unit Main;

interface

uses
    SysUtils, Windows, Messages, Classes, Graphics,
    Controls, Forms, Dialogs, StdCtrls, Buttons;

type
  TMainForm = class(TForm)
      ListBox1: TListBox;
        FindDialog: TFindDialog;
        FindBitBtn: TBitBtn;
        CloseBitBtn: TBitBtn;
        ReplaceBitBtn: TBitBtn;
        ReplaceDialog: TReplaceDialog;
        procedure FindBitBtnClick(Sender: TObject);
        procedure FindDialogFind(Sender: TObject);
        procedure ReplaceBitBtnClick(Sender: TObject);
        procedure ReplaceDialogFind(Sender: TObject);
        procedure ReplaceDialogReplace(Sender: TObject);
    private
      FindIndex, FoundPos, FoundLen: Integer;
        FoundItem: Boolean;
      public
    end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}

{ Begin a FindDialog operation }
```

```
procedure TMainForm.FindBitBtnClick(Sender: TObject);
begin
 FindDialog.Execute;
    FindIndex := 0;
    ListBox1.ItemIndex := -1;
end;

{ Continue a FindDialog operation }
procedure TMainForm.FindDialogFind(Sender: TObject);
var
    S: String;
begin
  while FindIndex < ListBox1.Items.Count do
     begin
       S := ListBox1.Items[FindIndex];
          Inc(FindIndex);
          if Pos(FindDialog.FindText, S) <> 0 then
          begin
           ListBox1.ItemIndex := FindIndex - 1;
              Exit;
             end;
       end;
     ShowMessage('No more matches!');
     FindDialog.CloseDialog;
end;

{ Start a ReplaceDialog operation }
procedure TMainForm.ReplaceBitBtnClick(Sender: TObject);
begin
     ReplaceDialog.Execute;
     FindIndex := 0;
     ListBox1.ItemIndex := - 1;
     FoundItem := False;
end;

{ Continue a ReplaceDialog operation }
```

```
procedure TMainForm.ReplaceDialogFind(Sender: TObject);
var
  S: String;
begin
  while FindIndex < ListBox1.Items.Count do
    begin
      S := ListBox1.Items[FindIndex];
        Inc(FindIndex);
        FoundPos := Pos(ReplaceDialog.FindText, S);
        if FoundPos <> 0 then
        begin
      ListBox1.ItemIndex := FindIndex - 1;
            FoundLen := Length(ReplaceDialog.FindText);
            FoundItem := True;
            Exit;
          end;
    end;
    ShowMessage('No more matches!');
    ReplaceDialog.CloseDialog;
end;

{ Perform replacement for a ReplaceDialog operation }
procedure TMainForm.ReplaceDialogReplace(Sender: TObject);
var
 S: String;
begin
 if frReplaceAll in ReplaceDialog.Options then
     ShowMessage('Replace All not implemented')
       else if not FoundItem then
     ShowMessage('Click Find to begin/continue search')
       else begin
     S := ListBox1.Items[FindIndex - 1];
       Delete(S, FoundPos, FoundLen);
       Insert(ReplaceDialog.ReplaceText, S, FoundPos);
       ListBox1.Items[FindIndex - 1] := S;
       FoundItem := False;
```

```
    end;
  end;

  end.
```

ان برمجةالـ Find هى اسهل المهتمين . أولاً ادخل event handler لأمر أو زر يبـدأ البـحـث . اسـتـدع الـ Execute method للـ FindDialog وقم باعـداد متـغيـرات عـمـومية تسـتـخـدم لاستكمـال البـحث . على سـبيل المثـال ، يبـدأ الـ FindRepl البحث بهذه العبارات :

```
FindDialog.Execute;
FindIndex := 0;
ListBox1.ItemIndex := -1;
```

يقوم السطر الاول باحضار modeless dialog . ويقوم السطر الثانى بإنشاء متغـير عـام يسـتـخدم للإشـارة لسطر الـ ListBox الذى تم بحـثه مؤخـراً . والسطر الأخـيـر يحـدد الـ ItemIndex فى الـ ListBox بـ (1-) بحيث لا يكون هناك اسطر يتم إبرازها من البداية .

عندمـا تبـدأ البـحث بضـغط زر الـ Find Next للـ dialog ، يقـوم الـ FindDialogFind بأداء البحث الفعلى باستخدام الـ FindIndex العام لتحديد أى من عناصر الـ ListBox يختبر . تقوم الـ Pos باختبار ما اذا كانت خاصية الـ FindText الموجودة فى الـ string المستهدف S ، منسوخة من الـ Items الخاصة بالـ ListBox . اذا كانت الـ Pos غير صفرية ، يوجد تجانس ، والتعيين للـ ItemIndex يبرز العنصر المتجانس .

لاحظ ان event handler يوجـد بعـد إيجـاد التـجـانس . هذا لا يغلق الـ dialog - إنه فقط يعيد الـ control الى البرنامج بحيث يمكن للمستخدمين من اداء اعـمـال اخـرى . ولكن ، اذا كـان الـ FindIndex العـام يسـاوى عـدد العناصـر ، يستدعى البرنامج الـ ShowMessage ليخبر المستخدمون ان البحث قد انتهى . يقوم الـ CloseDialog عندئذ باخفاء نافذة الـ FindDialog .

ان كتابة code لأمر الابدال يعتبر اصعب . إبدأ البحث كما مع الـ Find ، ولكن ابدأ أى مـتـغـيـرات عـامـة تحـتـاجـهـا لعـمل الابدال . (انظر الـ

ReplaceBitBtnClick فى القائمة). انك تحتاج الى Boolean Flag، مثل الـ FoundItem، للإشارة الى ان بنداً قد تم إيجاده. إبدأ هذا المتغير بـ False فى بداية البحث. وهنا يظهر كيف يستجيب البرنامج لضغط زر الReplace:

```
ReplaceDialog.Execute;
FindIndex := 0;
ListBox1.ItemIndex := -1;
FoundItem := False;
```

يقوم السطر الأول باستدعاء الـ Execute لعرض .modeless dialog box. يعين السطر الثانى صفراً لـ FindIndex والذى يمثل سطر الـ ListBox الذى يتم بحثه. يعين السطر الثالث (1-) لخاصية الـ ItemIndex للـ ListBox بحيث لا يتم ابراز اية اسطر. يبدأ السطر الأخير بالـ Boolean flag، (FoundItem)، بـ False.

لإنهاء امر الـ Replace، إنك تحتاج الى اثنين من event handler. الأول (وهو ReplaceDialogFind فى القائمة) يشبه للـ FindDialog event handler. عند إيجاد عنصر متجانس، ابدأ المتغيرات العامة التى تحتاجها لأداء إبدال. حدد FoundItem flag، بـ True بحيث يمكن الـ event handler الثانى ان يحدد أنه تم استدعاؤه كنتيجة للبحث المتجانس.

فى هذا event handler (انظر ReplaceDialogFind)، اذا كان الـ FoundItem، بـ True، ينفذ الـ procedure عبارات الإبدال الخاصة به. كما توضح القائمة، يمكنك ايضاً فحص ما اذا كان المستخدمون قد ضغطوا زر الـ Replace All باختيار ما اذا كان ثابت الـ frReplaceAll موجود فى مجموعة الـ Options للـ dialog. بالطبع، يعتبر code الابدال الفعلى فريداً لبرنامجك، ولكن Delphi لا يستطيع ان يفعل كل شئ (أليس كذلك؟).

: Page Controls

ان ثلاثة components جديدة نسبياً قد يسرت مهمة إنشاء dialogs ونوافذ اخرى متعددة الصفحات وهذه الـcomponents الثلاثة هى:

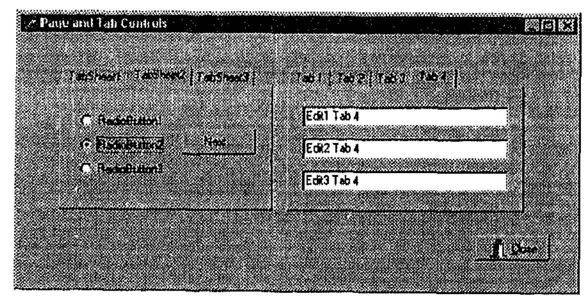
- **PageControl**: عندما تحتاج الى إنشاء نافذة أو dialogs متعدد الصفحات بمجموعات منفصلة من الـ controls فى كل صفحة، فإن هذا الـ component هو افضل اختيار.

- **TabSheet**: تعتبر كل صفحة فى نافذة أو الـ PageControl dialog هو عبارة عن TabSheet . هذا الـcomponent غير موجود على الـ VCL palette- انك تنشيء الـ TabSheet objects بالضغط على الزر الأيمن للفأرة داخل الـ PageControl واختيار امر الـ New Page من القائمة الناتجة .

- **TabControl**: عندما تحتاج dialog أكثر بساطة ، single-component ، متعدد الصفحات ، استخدم هذا الـ component بدلاً من الـ PageControl و الـ TabSheet . ان النتائج النهائية تعتبر متشابهة بصرياً، ولكن وضع الـ component على الصفحات المنفردة يعتبر مسئوليتك . تعتبر الـ TabControls مفيدة فى انشاء نوافذ متعددة الصفحات بنفس الـ controls فى كل صفحة، ولكن بمحتويات مختلفة- مجموعة من الـ Edit، مثلاً، التى تعتبر محتوى نصها اعتماداً على أى صفحة يختار المستخدم.

توضح الفصول التالية كيفية استخدام كلاً من الـ components الثلاثة . ان الـ PageControl والـ TabControl موجودان على الـ Win32 palette يتم إنشاء الـ TabSheet باستخدام الـ PageControl كما ستضح فى الفصل التالى .

على القرص المدمج: يوضح شكل (١٢-٧) أمثلة من الـPageControl والـ TabControl . ان الصفحات الموجودة فى الـ PageControl تعتبر الـ TabSheet . هذا الـ objects يوضح مظهر الـ PageControl، الموجود فى دليل الـ Source\PageTab على القرص المدمج . توضح القائمة (١٢-٩) الـ source code لـ PageTab.



شكل (١٢-٧): برنامج العرض PageTab يوضح نماذج من الـ PageControl (يساراً) والـ TabControl (يميناً). كل صفحة من الـ PageControl تعتبر TabSheet

القائمة (١٢-٩): PageTab\Main.pas

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
    Forms, Dialogs, StdCtrls, ComCtrls, Buttons;

type
  TMainForm = class(TForm)
      PageControl1: TPageControl;
        TabSheet1: TTabSheet;
        TabSheet2: TTabSheet;
        CheckBox1: TCheckBox;
        CheckBox2: TCheckBox;
        CheckBox3: TCheckBox;
        RadioButton1: TRadioButton;
        RadioButton2: TRadioButton;
        TabControl1: TTabControl;
        Edit1: TEdit;
        Edit2: TEdit;
        Edit3: TEdit;
        BitBtn1: TBitBtn;
        TabSheet3: TTabSheet;
        DateTimePicker1: TDateTimePicker;
        Button2: TButton;
        RadioButton3: TRadioButton;
        Button1: TButton;
        procedure TabControl1Change(Sender: TObject);
        procedure Button1Click(Sender: TObject);
        procedure Button2Click(Sender: TObject);
      private
      { Private declarations }
      public
      { Public declarations }
```

```
  end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}

procedure TMainForm.TabControl1Change(Sender: TObject);
var
  S: String;
begin
    S := IntToStr(TabControl1.TabIndex + 1);
    Edit1.Text := 'Edit1 Tab ' + S;
    Edit2.Text := 'Edit2 Tab ' + S;
    Edit3.Text := 'Edit3 Tab ' + S;
end;

procedure TMainForm.Button1Click(Sender: TObject);
begin
  PageControl1.ActivePage := TabSheet3;
end;

procedure TMainForm.Button2Click(Sender: TObject);
begin
  PageControl1.ActivePage := TabSheet1;
end;

end.
```

الـ PageControl component :

يبسط الـPageControl عملية إنشاء dialogs متعددة الصفحات ذات مجموعات فريدة من الـ controls على كل صفحة . لترى مثالاً على الـPageControl اثناء عمله ، قم بتحميل الـ PageTab.dpr فى Delphi من دليل

الـ Source\PageTab . يعــرض البــرنامج الـPageControl (يســاراً) والـ TabControl (يميناً). قبل تشــغيل البــرنامج لاحظ انه بامكانك اخــتيــار الـ PageControl tabs فى وقت التصـميم، ولكن لا يمكنك ان تفعل هـذا مع الـ TabControl . . وهذا لان صـفـحـات الـPageControl تعـتـبر TabSheet ، ولذلك، يمكن اختيارها وتعديلها باستخدام الـ Object Inspector . ولكن الـ TabControl يعتبر object واحـد- ان صفحاته ما هى الا خداع- ولذلك، لا يمكنك اختيار الـtabs المنفردة فى وقت التصميم.

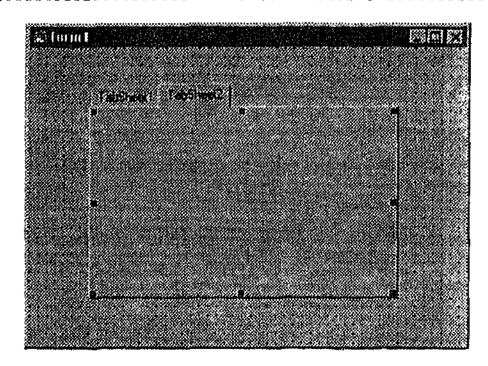
حاول تجربة الخطوات التالية لتعرف المزيد عن استخدام الـPageControl:

١- ابدأ تطبيقاً جديداً. اختر الـ Win32 page tab على الـVCL palette لـ Delphi واضغط الـPageControl (الثانى من اليسار). اضـغط داخل الـ from الـ PageControl . يبدو الـobject فى البداية شبه اللوحة الخالية.

٢- لإنشاء صفحات فى الـ PageControl، اضـغط زر الفأرة الأيمن داخل الـ PageControl . اختــر New Page من القـائمـة الناتجـة. هذا يؤدى الى إنشـاء TabSheet، أضف ذلك الـ object الى أى objects اخرى مملوكة للـ PageControl هذا، وعرض two pages . يوضح شكل (١٢-٨) العرض بعد إضافة صفحتين.

٣- اضــغط page tabs ولاحظ ان نافـذة الـ Object Inspector تعــرض خصائص للـ PageControl1 . اضغط داخل نافذة الـ PageControl (تحت page tabs) لاخـتـيـار TabSheet للصـفـحـة. يظهـر الـ Object Inspector عندئذ خصائص الـ TabSheet.

٤- لإضافة control لصفحة الـ PageControl، اختر الصـفحة ثم إسقط أى control عليـهــا. يمكنك ان تفـعل هـذا مع أى control تقــريبــاً، مــثل الـ Buttons، CheckBoxes ، الــ Labels ، الــ StringGrids ، والــ DateTimePickers . بالرغم من ان الـ controls تسـتـقـر على صـفـحــات الـ TabSheet المنفردة، فإنك تتوصل إليها بنفس طريقة وصولك الى أى controls بالإشارة الى اسمائها مثل Button1 و CheckBox3 . لا يجب عليك ان تستخدم الـ PageControl objectللوصول الى الـ controls التى يحتويها.



شكل (١٢-٨): الـ PageControl component مع اثنين من الـ TabSheet

خصائص الـ PageControl:

يوفر الـ PageControl خصائص يمكنك استخدامها للتعامل مع الـ controls بطرق متنوعة . باستخدام الـ PageControl من الفصل السابق ويمكنك اسقاط واحداً على الـ from الخالية)، قم بتجربة مواصفات الخصائص التالية :

- **ActivePage:** حدد هذه باسم الـ TabSheet الذى تريد عرضة على انه الصفحة النشطة الأولى . فى وقت التشغيل، يمكن لعبارة program ايضاً ان تعين هذه القيمة لتغيير الصفحة النشطة تحت تحكم البرنامج [انظر الـ Button2Click procedure فى القائمة (١٢-٩) كمثال].

- **DockSite:** حددة بـ True لإنشاء PageControl يعمل كـ docking station، للـ controls الاخرى. انظر "إنشاء الـ Docking Controls" فى هذا الباب لمزيد من المعلومات حول إنشاء docking sites، وهى ميزة جديدة فى Delphi 4 .

- **Hint:** إدخل نصاً للعرض. عندما يبقى المستخدم مؤشرة الفأرة على الـ controls للحظة أو إثنين. استخدم هذه الخاصية مع الـ ShowHint لعرض نفس نص الملحوظة بغض النظر عن الصفحة النشطة. انظر خصائص الـ Hint والـ ShowHint للـ TabSheet لمزيد من المعلومات.

- **HotTrack:** حددها بـ True لتوفير تغذية خلفية بصرية. عندما يمرر المستخدم مؤشرة الفأرة على الـ page tab، يتم إختفاء نصها لوقت قصير ليرى المستخدمون ان بامكانهم ضغط الفأرة لاختيار ذلك الـ tab .

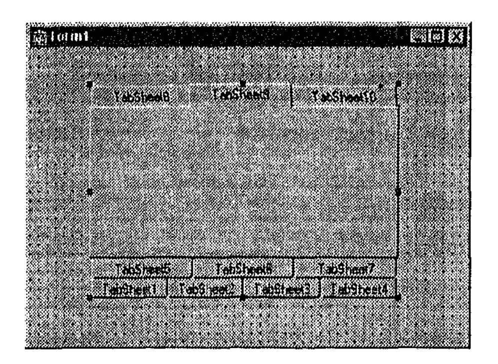
• **MultiLine:** فى الـ PageControls المعقدة ذات الصفحات الكثيرة، حدد هذه الخاصية بـ True لعرض الصفحات على صفين أو أكثر اذا لم تكن جميعها تتناسب على صف واحد بداخل عرض النافذة المحددة. عندما تكون هذه الخاصية محددة بـ False، تظهر الـ Scroll arrows اذا كان هناك صفحات كثيرة جداً بحيث يصعب ان تتناسب على صف واحد- ولكن، الصفوف المتعددة تعتبر افضل فى معظم الحالات.

• **MultiSelect:** هذه خاصية غامضة تعمل فقط عندما يكون نمط الـ tab محدد بـ Button أو tsFlatButton. إنها تبدو كسمة من سمات الـ TabControl يتم عرضها فى الـPage control، ولكن اختيار صفحات متعددة لا يبدو كعملية ذات قيمة عملية كبيرة. إنها غير مؤقتة، ولا يبدوان لها تأثيراً كبيراً، إن وجد أصلاً. مسمى هذه الخاصية باللغز.

• **OwnerDraw:** عندما يتم تحديد هذه الخاصية بـ True، فإنها مسؤليتك أن تعرض نصاً أو رسوماً جرافيكية (أو كلاهما) فى كل tab. لتفعل هذا، قم بإنشاء OnDrawTab event handler وأدخل code لعـــرض نص أو bitmaps فى مساحة page tab المخططة بواسطة Rect parameter. أنظر «إنشاء الـ owner draw PageControls» فى هذا الباب للحصول على مزيد من المعلومات.

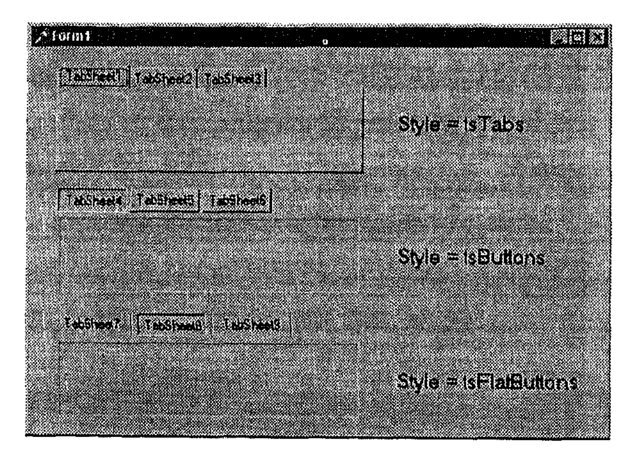
• **ScrollOpposite:** عندما يكون للـ PageControl tabs عديدة، يمكنك تحديد الـ ScrollOpposite بـ True أو False لتغيير الطريقة التى تتحرك بها tabs أخرى، بعيداً عن الطريق عندما يختار المستخدمون tab معيناً. على سبيل المثال، فى PageControl ذى صـــفين من ثلاث tabs، اذا كـــان الـ ScrollOpposite محدد بـ True، فإن اختيار tab فى إحدى الصفوف يحرك الصف الآخر الى أسفل العرض [أنظر شكل (١٢-٩)].

• **ShowHint:** حدد هذه الخاصية بـ True لعرض النص الموجود فى خاصية الـ Hint عندما يبقى المستخدم مؤشر الفأرة على الـ PageControl للحظة أو إثنين. لعرض نص مختلف إعتماداً على أى صفحة تكون نشطة، حدد الـ ShowHint بـ False واستخدم خصائص الـ Hint والـ ShowHint فى الـ TabSheet (أنظر الفصلين التاليين).



شكل (١٢-٩): تحديد الـ ScrollOpposite للـ PageControl بـ True يؤدى الى Scroll tabs المتعددة الى النهاية المعاكسة للـ control عندما يختار المستخدمون tab فى مجموعة أخرى

• **Style:** هذه الخاصية تؤثر على طريقة عرض الـ tabs [شكل (١٢-١٠) يوضح نماذج لكل صفة) حدد الـ tsTabs للـ control الذى يبدو مثل page tabs على folders الملفات؛ استخدم الـ tsButtons لعرض page tabs على ازرار الـ Windows ؛ استخدم tsFlatButtons لعرض page tabs دون ايه حدود، ولكن عند اختيارها. (فكرة: عند استخدام نمط الـ tsFlatButtons، يجب ان تحدد الـ Hottrack بـ True لعرض تغذية خلفية فى الـ form للأزرار المتبقية عندما يمرر المستخدم مؤشرة الفأرة فوق control page tabs).



شكل (١٢-١٠): استخدم خاصية الـ PageControl Style لتؤثر على طريقة عرض الـ tabs

- **TabPosition**: يمكنك استخدام هذه الخاصية لاختيار ما اذا كنت ستعرض page tabs فى أعلى الـ control (tpTop) أو اسفله (tpBottom) . يتم عرض عادة فى أعلى الـ Control وكثير من المستخدمين يتوقعون ان يجدوها فى هذا الموضع . من الأفضل الا تغير هذه الخاصية الا اذا كان لديك سبب وجيه .

Tip

فكرة: عندما تكون خاصية الـ Style للـ PageControl محدد بـ tsButtons أو tsFlatButtons ، لا يظهر أى border حول الصفحات المنفردة للـ control . اذا كنت تريد حداً ، إدخل Panel فى كل صفحة ثم قم باسقاط الـ controls على الـ Panels . أو احفظ لـ handle الـ Window واستخدم Bevel بدلاً منه .

الـ TabSheet component:

كل صفحة على الـ PageControl تعتبر فى حد ذاتها object من الـ TTabSheet class . كما ذكرت ، الطريقة الوحيدة لإنشاء TabSheets هى بالضغط على الزر الأيمن للفأرة فى الـ PageControl واختيار امر الـ New Page من القائمة . لاختيار الـ TabSheet فى الـ PageControl ، اضغط الـ page tab الذى تريده ، ثم اضغط داخل control body (تحت page tabs اذا كانوا على الـ control) . يمكنك عندئذ استخدام الـ Object Inspector لعمل تغييرات لخصائص الـ TabSheets المنفردة .

Tip

فكرة: عند تحديد خصائص لـ TabSheets متعددة ، من الصعب ، ولكن ممكن ، أن تختارها جميعاً فى نفس الوقت . لتفعل ذلك ، استمر فى ضغط مفتاح الـ Shift أثناء الضغط داخل كل page tab ، ثم فى كل TabSheet (فى con-trol body) . بعد اختيار الـ TabSheet النهائى ، بينما مازلت تضغط الـ Shift ، اضغط أى page tab لإبطال اختيار الـ PageControl نفسه . يمكنك عندئذ تعيين قيم خاصية لكل الـ TabSheets المختار ـ حدد خاصية الـ Cursor بـ crCross مثلاً .

خصائص الـ TabSheet:

فيما يلى خصائص TabSheet مختارة والتى قد تريد تغييرها عند تصميم الـ PageControl . لتجربة هذه المواصفات ، قم بإسقاط PageControl على

form، اضغط على الزر الأيمن للفأرة داخل الـ control، واختر New Page لإضافة صفحة واحدة أو أكثر . اختر صفحة الـ TabSheet بالضغط داخل الـ PageControl (تحت الـ PageControl)، قم بتجربة مواصفات الخصائص هذه :

- **BorderWidth:** حددها بقيمة موجبة غير صفرية، لتغيير عرض الـ Border حول صفحة الـ TabSheet هذه .

- **Caption:** اكتب أى نص تريده ليظهر فى tab هذه الصفحة . يتم تعديل عرض tab بصورة تلقائية ليسع النص . اذا كان مجال الـ OwnerDraw التابع لـ PageControl محدد بـ True، فإنها مسئوليتك أن تعرض نص .

- **Cursor:** اختر واحداً من الـ cursor المذكورة فى قائمة اللائحة التابعة لهذه الخاصية . تتغير الـ cursor بصورة تلقائية عندما تمر على الصفحة .

- **Hint:** أضف الـ hint text ليظهر فى موضع الـ cursor عندما تستقر للحظة أو إثنيتين . حدد الـ ShowHint بـ True لعرض نص الـ Hint هذا .

- **PageIndex:** هذا الحقل محدد بصورة تلقائية بقيمة الفهرس لكل صفحة TabSheet . والفهرس للصفحة الأولى هو صفر، والتالية واحد، الى آخره . اذا قمت بتغيير هذه القيمة فى وقت التصميم، يتم تعديل الصفحات الأخرى طبقاً لذلك، ويتم إدخال الصفحة التى تغيرت فى موضع جديد . يمكنك فعل هذا بعد إنشاء الـ PageControl لتغيير نظام صفحاته المنفردة، ولكن فى أغلب الحالات لن تحتاج الى إدخال قيم فى هذه الخاصية .

Note

ملحوظة: تشير online help الى أن الـ PageIndex محدد بـ (-1) عندما يكون الـ TabVisible محدد بـ False . والأمر ليس كذلك حيث تبقى قيمة الـ PageIndex كما هى بغض النظر عما اذا كانت صفحة الـ TabSheet مرئية أم لا .

- **PopupMenu:** أضف PopupMenu من الـ Standard palette على الـ form، يمكن للمستخدمين عندئذ أن يضغطوا يمين الفأرة أثناء الإشارة بالـ cursor داخل صفحة الـ TabSheet لعرض قائمة . يمكن لكل الصفحات أن تستخدم نفس

الـ PopupMenu، أو يمكن أن يكون لكل صفحة object مختلفاً. هذه التقنية تعتبر طريقة عظيمة لتوفير مجموعات مختلفة من الأوامر المحلية إعتماداً على الصفحة النشطة.

● **ShowHint:** حددها بـ True لعرض النص الذى تم إدخاله فى خاصية الـ Hint عندما يبقى المستخدم مؤشرة الفأرة على هذه الصفحة لعدة ثوان. استخدم هذه الخاصية وخاصية الـ Hint لعرض hint فريدة إعتماداً على الصفحة النشطة.

Tip

فكرة: لعرض نفس الـ hint لكل صفحات الـ TabSheet، من السهل أن تحدد الـ ShowHint بـ False، والـ ParentShowHint بـ True. اختر الـ PageControl، أدخل نص الـ Hint الخاص بك، وحدد خاصية الـ ShowHint لهذا الـ control بـ True.

● **TabVisible:** حدد هذه الخاصية بـ False فى وقت التشغيل حتى تزيل مؤقتاً الـ tab sheet من الصفحة. هذا أيضاً يحدد خاصية الـ TabIndex للـ Sheet بـ (-1). إن إعادة تحديد الـ TabVisible بـ True مرة أخرى يظهر الـ tab sheet.

Tip

فكرة: عندما يكون الـ TabVisible محدد بـ False، لم يعد من الممكن إختيار الـ TabSheet باستخدام الفأرة لأن تلك الصفحة لم تعد معروضة. بدلاً من ذلك، اختر هذه الـ TabSheet من قائمة اللائحة للـ Object Inspector. يمكنك عندئذ إعادة تحديد الـ TabVisible بـ True لتجعل الصفحة مرئية مرة أخرى.

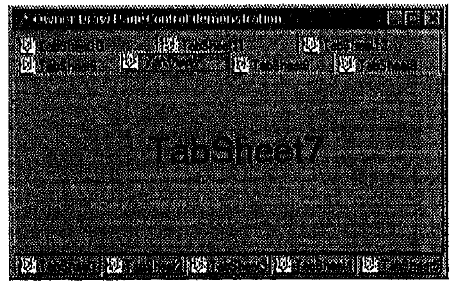
إنشاء PageControls خاصة بالـ owner-draw:

لكى تخصص عرض محتويات page tab، يمكنك إنشاء PageControl خاص بالـ owner-draw. هذا يضع مسئولية عرض النص أو الرسوم الجرافيكية للـ page tab على عاتقك- وهى ميزة قد تستخدمها، مثلاً، لعرض page tabs باستخدام fonts مختلفة، أو لعرض أيقونة.



على القرص المدمج: يوضح شكل (١٢-١١) مثالاً على الـ PageControl الخاص بالـ owner-draw، والذى أضفت له ١٢ صفحة. يعرض البرنامج نص page tab النشطة بالخط المائل، ويعرض

أيضاً أيقونة صغيرة بعد كل tab label . توضح القائمة (١٢-١٠) الـ source code لهذا البرنامج- وهى موجودة على القرص المدمج فى دليل الـ Source\PageTabOD . لتوفير المساحة ، قمت بحذف بضعة أسطر تقوم بمجرد تعريف الـ Label والـ TabSheet فى الـ form class- يقوم Delphi بإنشائهم ويمكنك تجاهلها .



شكل (١٢-١١): إنه باستخدام الـ owner-draw PageControl، يمكنك أن تعرض نص الـ page tab باستخدام font مختلف (هذا المثال يستخدم الخط المائل للصفحة النشطة)، ورسوم جرافيك مثل الأيقونات الموضحة هنا

القائمة (١٢-١٠): PageTabOD\Main.pas

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
     Forms, Dialogs, ComCtrls, Images, StdCtrls;

type
  TMainForm = class(TForm)
       PageControl1: TPageControl;
          TabSheet1: TTabSheet;
    // TabSheet2 ... TabSheet 12 deleted
       ImageList1: TImageList;
          Label1: TLabel;
    // Label2 ... Label12 deleted
```

```
        procedure PageControl1DrawTab(Control: TCustomTabControl;
          TabIndex: Integer; const Rect: TRect; Active: Boolean);
            private
        { Private declarations }
          public
        { Public declarations }
          end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}

procedure TMainForm.PageControl1DrawTab(
    Control: TCustomTabControl; TabIndex: Integer;
    const Rect: TRect; Active: Boolean);
begin
    with Control.Canvas do
    begin
     if active
          then Font.Style := [fsItalic]
            else Font.Style := [];
           TextRect(Rect, Rect.Left + 20, Rect.Top + 3,
          PageControl1.Pages[TabIndex].Caption);
           ImageList1.Draw(Control.Canvas,
             Rect.Left + 2, Rect.Top + 2, 0);
    end;
end;

end.
```

اتبع هذه الخطوات لإنشاء برنامج العرض PageTabOD المذكور هنا :

١ - إبدأ تطبيقاً جديداً.

٢- قم باسقاط الـ ImageList من الـ Win32 على الـ form . اضغط مرتين الـ component واضغط زر الـ Add لإضافة صورة أيقونة . لقد استخدمت أيقونة الـ Sample.ico من دليل الـ Source\Data على القرص المدمج ، ولكن بإمكانك أن تستخدم أى صورة أخرى تريدها .

٣- قم باسقاط الـ PageControl من الـ Win32 على الـ form . اضغط زر الفأرة الأيمن داخل الـ component واختر New Page من القائمة . افعل هذا ١٢ مرة .

٤- إختيارى : قم باسقاط الـLabel على كل صفحة من الـ PageControl . لتفعل هذا ، اختر كل tab واضغط داخل الـ PageControl هذا يؤدى الى اختيار الـ TabSheet الذى يعمل كوعاء لـ controls هذه الصفحة (افحص نافذة الـ Object Inspector لتتأكد من أنك قد اخترت الـ TabSheet) ثم ، قم باسقاط الـ Label على الـ TabSheet . هذا يربط الـ label بالصفحة- تظهر الـ label عندما تكون الصفحة الخاصة بها نشطة فقط . استخدم نفس الطريقة لإسقاط أى أنواع controls أخرى على صفحات الـ PageControl .

٥- اختر PageControl1 . يمكنك أن تفعل هذا إما باستخدام قائمة إسقاط الـ Object Inspector أو بضغط أى page tab . تأكد من أن الـ Object Inspector يذكر الـ PageControl1 فى البداية .

٦- حدد الـ Align بالـ alClient حتى يملأ الـ PageControl الـ client area بالنافذة . هذه الخطوة إختيارية ، ولكن عندما تنشئ Pages dialogs ، فإنك غالباً ما تختار هذا التحديد .

٧- حدد الـ MultiLine بـ True حتى تصبح كل page tabs مرئية فى صفوف متعددة . وكذلك حدد الـ ScrollOpposite بـ True بحيث تتحرك بعض الصفوف الى أسفل عند الحد السفلى للنافذة .

٨- حدد الـ OwnerDraw بـ True . عند هذه النقطة ، تصبح page tab labels فارغة .

٩- لعرض الأيقونات والنص فى كل page tab ، يجب أن تنشئ الـ event handler لـ PageControl . تأكد من أن الـ PageControl1 مازال مختاراً ، ثم

اضـــغط الـ Events tab في الـ Object Inspector . اضـــغط مـــرتين الـ OnDrawTab لإنشاء procedure، والذى يمكنك أن تملأه بالـ code الموجودة فى القائمة.

عندما تقوم بتشغيل البرنامج التام بضغط F9، يتم استدعاء OnDrawTab لرسم النص والرسوم الجرافيكية لكل Page tab . يتم تعريف الـ procedure والـ parameters الخاصة به كما يلى :

```
procedure TMainForm.PageControl1DrawTab(
  Control: TCustomTabControl; TabIndex: Integer;
    const Rect: TRect; Active: Boolean);
```

- Control: هذا يشير الى الـ object الذى يكون صورة الـ tab . لكى ترسم داخل هذه المساحة، استخدم خاصية الـ Canvas للـ Control .
- TabIndex: هذا يساوى الـ TabSheet Index الذى يكون الصـفـحة، بصفر للصفحة الأولى، واحد للثانية، وهكذا.
- Rect: القيم الموجودة فى هذا السجل تحدد الأحداثيات المتعلقة للرسم فى الـ tab . غـالبـاً، يمكنك اسـتـدعـاء الـ methods فى الـ Canvas وتمرير الـ Rect أو قيمة داخلية مثل الـ Rect.Left والـ Rect.Top على أنها arguments .
- Active: يكون محدد بـ True اذا كان tab للصفحة الذى يتم رسمه هو للصفحة النشطة (page tab الذى يتم إختياره فى وقت التشغيل). اذا كان محدد بـ False، يكون الـ procedure قد رسم صفحة غير نشطة- ألا وهى، صفحة ذات controls غير مرئية.

يستخدم البرنامج هذه الـ parameters لرسم أيقـونات ونص فى كل page tab . من خلال عبارة الـ with نلاحظ أننا نستخدم خاصية الـ Canvas المرتبطة بالـ page tab وليس لـ object آخر :

```
with Control.Canvas do
begin
  // Call methods for the Control Canvas
end;
```

والخطوة الأولى هى لتحديد ما اذا كنا نرسم الصفحة النشطة. اذا كان كذلك، يحدد البرنامج نمط الـ Canvas font بالخط المائل؛ وإن لم يكن كذلك، يتم تحديد النمط بالـ font العادى (لا ثوابت نمط معين)، باستخدام عبارة الـ if:

```
if active
  then Font.Style := [fsItalic]
    else Font.Style := [ ];
```

لعرض تعليقات page tab، يستدعى البرنامج الـ TextRect method فى Canvas:

```
TextRect(Rect, Rect.Left + 20, Rect.Top + 3,
  PageControl1.Pages[TabIndex].Caption);
```

الـ arguments الثلاث الأولى للـ TextRect تحدد الـ Rect التام، الذى يعرف الحدود التى يجب أن يظهر فيها النص، والإحداثيات اليسرى والعلوية للـ pixel الأيسر العلوى للنص. والـ argument الأخيرة هى النص الذى يجب عرضه. وهذا قد تم أخذه من الـ Pages array لـ PageControl، باستخدام قيمة الـ TabIndex الذى تم تمريرها لـ OnDrawTab. هذا يحدد موضع الـ TabSheet الذى يمثل الصفحة وخاصية الـ Caption له.

لرسم الأيقونات يستدعى البرنامج الـ Draw method فى الـ ImageList الذى يحمل صورة الأيقونة:

```
ImageList1.Draw(Control.Canvas,
  Rect.Left + 2, Rect.Top + 2, 0);
```

تحدد الـ argument الأولى أن الـ Canvas الخاص بـ page tab يستخدم فى رسم الصورة. و الـ arguments الآخرتان تحددان موضع الصورة. والـ argument الأخيرة هى Index الصورة فى الـ ImageList. بالرغم من أن التطبيق يرسم نفس الأيقونة فى كل page tab، إلا أنه من السهل عرض أيقونات مختلفة. قم بتحميل صور الأيقونات فى الـ ImageList الخاص بك، وحدد قيم الفهرس الخاصة بها بـ ImageList.Draw. (فكرة: يمكنك استخدام الـ TabIndex parameter الذى تم تمريره لـ OnDrawTab- بهذه الطريقة، تصبح قيم فهرس الصورة تماماً مثل الـ TabSheet فى الـ PageControl).

Note

ملحوظة: إن عروض page tabs يتم تحديدها بصورة تلقائية بكم النص الذى يعرضه OnDrawTab .

الـTabControl component:

إن الطريقة السهلة لإنشاء controls dialogs أخرى متعددة الصفحة هى باستخدام الـ TabControl على الـ Win32 . وعلى عكس الـ PageControl الأكثر تعقيداً، إن الـ TabControl ليس لها صفحات منفصلة، بالرغم من أنها تبدو كذلك. إن الصفحات فى الـ TabControl هى مجرد خداع. يستطيع المستخدمون إختيار TabControl page tabs ، كما يفعلون مع الـ PageControl- من الناحية البصرية، يبدو الـ objects وكأنهما واحد. ولكن، لأن الـ TabControl ليس له صفحات حقيقية.

وإحدى الاستخدامات العملية للـ TabControl هى إنشاء forms إدخال بيانات ذات مجموعات من الـ controls تتغير محتوياتها إعتماداً على الـ tab المختار. على سبيل المثال، قم بتحميل ملف مشروع الـ PageTab.dpr فى Delphi- على القرص المدمج، يوجد هذا الملف فى دليل الـ Source\PageTab . والـ TabControl الموجود على اليمين [راجع شكل (١٢-٧)] يوضح ثلاثة Edit objects . قم بتشغيل البرنامج بضغط F9، واختر كلاً من page tabs الأربع تبقى الـ Edit controls كما هى، ولكن تتغير محتوياتها إعتماداً على الصفحة النشطة. ولكن تذكر أنه لا يوجد صفحات حقيقية فى هذه الحالة- page tabs فقط تغير مظهرها لتشير الى أى مساحة فى الـ control يتم إختيارها.

لإدخال نص فى الـ Edit، ينفذ البرنامج هذا الـ procedure :

```
procedure TMainForm.TabControl1Change(Sender: TObject);
var
  S: String;
begin
  S := IntToStr(TabControl1.TabIndex + 1);
    Edit1.Text := 'Edit1 Tab ' + S;
    Edit2.Text := 'Edit2 Tab ' + S;
    Edit3.Text := 'Edit3 Tab ' + S;
end;
```

هذا الـ code لأغراض العرض فقط، ولكنها تظهر كيفية تعديل الـ TabControl إعتماداً على أى page tab يتم إختياره. إن الـ OnChange للـ TabControl، والذى يتم استدعاؤه عندما يختار المستخدم page tab. لتحديد الـ tab الذى تم إختياره، استخدم خاصية الـ TabIndex للـ TabControl. الـ tab الأول يحمل رقم صفر، والثانى رقم واحد وهكذا. لترقيم الـ labels الموضحة فى الـ Edit، يضيف البرنامج واحد لهذه القيمة وينشئ string S لإضافته للـ static text الموضحة. وهذه يتم تعيينها لخصائص الـ Text للـ Edit. فى برنامج حقيقى، يمكنك استخدام تقنيات مشابهة لتغيير محتويات الـ controls- بقراءة معلومات النص من الملف، مثلاً.

Tip **فكرة:** فى وقت التصميم، يمكنك تغيير الـ TabIndex التابعة للـ TabControl لتنشيط tab مختلفاً. على عكس الـ PageControl، لا تعد الـ TabControl tabs قابلة للتعامل معها فى وقت التصميم.

خصائص الـ TabControl:

إن خصائص الـ TabControl تشبه خصائص الـ PageControl، لذا لن أذكرها منفصلة (أنظر خصائص الـ PageControl فى هذا الباب). إن الخاصية الوحيدة المختلفة هى الـ Tabs، وهى TStrings. اضغط قيمة هذه الخاصية مرتين، أو اضغط الزر البيضاوى لها، لفتح Delphi string-list editor. كل سطر نص فى هذا الـ editor ويمثل Page tab label. اكتب أى عدد من labels التى تحتاجها لإنشاء الـ TabControl page tabs.

تحديد حجم النافذة:

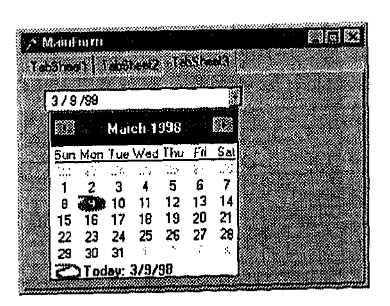
فى النسخ السابقة من الـDelphi لم يكن فى الإمكان تحديد حجم النافذة (أو على الأقل، لم يكن سهلاً). لقد كان هذا يؤدى عادة الى مشاكل عندما يقوم المستخدمون بإعادة تحديد حجم النافذة، مما يغطى بعض controls الخاص بالنافذة. والخيار الوحيد أن تجعل حجم النافذة ثابتاً، ولكن كان هذا معوق للغاية.

إن Delphi 4 يضيف خاصية جديدة لـ TControl class: وهى الـ TSizeConstraints. يمكنك استخدام هذه الخاصية لتحدد الحد الأقصى والأدنى

للعروض والإرتفاعات لأغلب الـ components المشتقة من الـ TControl . حتى نافذة الـ from يمكن أن تكون مقيدة الحجم .

يعد استخدام خاصية الـ Constraints أمراً بسيطاً . وكمثال ، ابدأ تطبيقاً جديداً وإبحث عن خاصية الـ Constraints فى نافذة الـ Object Inspector للـ Form1 للتطبيق اضغط مرتين علامة الزائد الصغيرة الى اليسار من الـ Constraints . هذا فتعرض القيم الخواص الفرعية الأربعة : MaxHeight ، MaxWidth ، MinHeight ، و MinWidth . إن هذه القيم تكون صفراً فى المعتاد ، مما يغلق كل الـ Constraints . أدخل أى أعداد صحيحة فى هذه الخصائص لتحديد النافذة أو إرتفاع وعرض أى control آخر . إن القيم هى أعداد صحيحة موجبة تمثل أحجام الـ control بالـ pixels .

وبرنامج العرض Constraint الموجود على القرص المدمج فى دليل الـ Source\Constraint يوضح كيفية إنشاء الـ PageControl . والهدف فى برنامج العرض هو ضمان أن المستخدمين لا يستطيعون تصغير نافذة dialog الى حد كبير بحيث تؤثر على controls أخرى مثل الـ CheckBoxes والـ RadioButtons . فى نفس الوقت ، نريد منع الـ dialog من أن يملأ الشاشة ، مما يهدد حالة الشاشة الفعلية وتبدو سيئة . يوضح شكل (١٢-١٢) عرض البرنامج- لاحظ أن الـ PageControl يملأ النافذة . ان الـ PageControl هو الذى تم تقيد حجمه ، ولكن لان النافذة تملك الـ control ، فهى الأخرى لهـا نفس القيود . ان ملف الـ Main.pas لهذا البرنامج ليس لهcode مهمة ، لذا اذكرها هنا .



شكل (١٢-١٢): ان الـ PageControl يقيد حجم نافذته، مما يضمن ان المستخدمين لا يستطيعون تصغير النافذة وتغطية الـ controls و تكبير النافذة اكبر من حجمها العملى.



Note

ملحوظة: أن ضغط زر الـ maximize يكبر النافذة لأقصى عرض وطول محدد لها ، وكذلك يضع النافذة إلى الركن الايسر العلوى من الشاشة- قد تريد إضافة ملحوظة فى توثيق تطبيقك حول تأثير احجام النافذة .

لكى تنشئ البرنامج بنفسك ، إتبع مايلى :

١- قم باسقاطPageControl على form خالية . اضغط زر الفأرة الايمن داخل الـ PageControl واختر New Page من القائمة لإضافة صفحات قليلة .

٢- قم باسقاط بعض الـ controls الاخرى فى صفحات الـ PageControl . لا يهم أى الـ controls تستخدم . على سبيل المثال ، أضف ثلاثة من الـ CheckBoxes الى الصفحة الأولى ، وثلاثة من الـ RadioButtons الى الصفحة الثانية ، و DateTimePicker الى الصفحة الثالثة .

٣- مازلنا فى الـ Delphi ، حدد حجم النافذة باقصى ما تريد . لاحظ قيم خصائص الـ Width والـHeight للـPageControl . (تأكد من ان الـ Object Inspector يعرض خصائص الـ PageControl1- اذا لم يكن ، اختر ذلك من القائمة الخاصة بالـObject Inspector) . اضغط مرتين خاصية الـ Constraints ، وإدخل قيم الـ Width والـHeight المشار إليها فى الـ MaxWidth والـ MaxHeight على التوال .

٤- قم بتصغير النافذة الى اصغير حجم يبدو جيداً ولا تغطى أى من الـcontrols التى اضفتها للـPageControl أعد الخطوة الثالثة ، ولكن هذه المرة إدخل قيم الـ Width والـHeight المشار إليها فى الـMaxHeight والـ MaxHeight .

٥- قم بتشغيل البرنامج بضغط الـ F9 . حاول ان تحدد حجم النافذة- فهى لا تصغر ولاتكبر عن الحدود المحددة . وكذلك ، اضغط زر الشاشة الكاملة ولاحظ ان الشاشة تكبر فقط الى اقصى حد للحجم المسموح به .

Caution

تحذير: ان إضافة controls الى حاوية مثل الـ PageControl يؤدى الى إعادة تحديد قيم خاصية الـ Constraints لها بصفر . قم ببرمجة خاصية الـ Constraints فى هذه الحالات فقط بعد إدخال كل الـ controls الصغيرة فى الحاوية الأم .

فى وقت التشغيل، يمكنك ان تتخطى قيود الـ controls باستخدام event handler جديد مخصص لهذا الغرض. قد تفعل هذا لتسمح للمستخدمين بتكبير نافذة إلى حجم الشاشة الكاملة.



على القرص المدمج: لتجربة الـ event handler هذا، أعد تحميل مشروع الـ Constraint.dpr من دليل الـ Source\Constraint على القرص المدمج الى Delphi، ثم إتبع هذه الخطوات:

١ - اختر MainForm باستخدام القائمة التابعة للـ Object Inspector.

٢ - اضغط الـ Events tab على نافذة الـ Object Inspector.

٣ - اضغط مرتين الـ OnConstrainedResize لانشاء event handler خالى. إملأ code هذا البرنامج باستخدام القائمة (١٢-١١).

٤ - قم بتشغيل البرنامج بضغط F9. اعد تحديد حجم النافذة- يمكن ان تكبر الآن الى مساحة الشاشة كاملة. كذلك حاول ان تضغط زر الشاشة الكاملة للنافذة، الذى يعمل الآن بشكل طبيعى.

القائمة (١٢-١١): إدخل الـ code من OnConstrainedResize هذا الى مشروع الـ Constraint لتوضح كيفية تخطى قيود النافذة

```
procedure TMainForm.FormConstrainedResize(Sender: TObject;
  var MinWidth, MinHeight, MaxWidth, MaxHeight: Integer);
begin
  MaxWidth := Screen.Width;
     MaxHeight := Screen.Height;
end;
```

هذا الـ event خاص للـ TPanel classes، وTScrollBox، وTPanel فقط. استخدمه لتغير فى وقت التشغيل قيم خاصية الـ Constraint التى تم إدخالها فى وقت التصميم. على سبيل المثال، كما هو موضح هنا، يمكنك تحديد قيم الـ MaxWidth والـ MaxHeight على التوال للـcontrols لخصائص الـ Width والـHeight للـ Screen object، مما يتيح للنافذة ان تتماثل فى الحجم حجم الشاشة التامة.

إنشاء الـ Docking Controls:

كما تعرف غالباً يمكنك وصل نوافذ Delphi 4 المتنوعة معاً. على سبيل المثال، اختر View|Project Manager ، ثم اضغط واسحب نافذة الى الـ code editor عند الحد الايمن أو الايسر. عندما يغير خط السحب الحجم، اطلق زر الفأرة وتتصل النافذة بالـ code editor. يعتبر هذا مفيداً بشكل خاص مع عروض التقنيات العلياً، حيث يوجد وفرة في الاماكن الى اليمين أو اليسار من مساحة تحرير النص. لفك وصل نافذة، اضغطها واسحبها حتى يتغير حد تمييزها مرة اخرى ليعود لحجمه الطبيعي، ثم اطلق الفأرة.

يمكنك ايضاً إنشاء docking controls الخاصة بك باستخدام Delphi- النسخ الأولية ليس بها هذه الميزة. لتفعل هذا، إنك تحتاج الى objects اساسين:

- الـ docking site object.
- الـ dockable control object.

و dockable control object يمكن ان يكون أى component له خاصية DockSite. هذا يشمل الـ Form، الـ Panel ، الـPageControl ، الـTabControl ، الـToolBar ، الـ CoolBar و components اخرى. يجب ان يكون .

الـ docking site object قادراً على ان يملك controls اخرى الـcomponents البسيطة مثل الـButtons والـCheckBoxes لايمكن ان تكون docking sites .

و dockable control object هو أى component له خاصتين هما : الـDragKind والـDragMode. قليل من الـcomponents فقط هى التى لها DragMode- ولكن لكى تكون dockable، يجب ان يملك الـ control الخاصيتين. وامثلة الـcomponents التى لا يمكن ان تكون dockable هى تلك الموجودة على الـWin3.1 ، وتلك التى ليس لها أى شكل معين عند إسقاطها على الـ form مثل الـ Timer و الـ MainMenu.

Tip **فكرة:** ان اغلب التطبيقات تستخدم الـ Panel كـ docking sites. ان الـ Dockable controls هى تلك التى تحمل الـ controls الأخرى مثل الـ ToolBars، الـ CoolBars ، والـ PageControls .

ان النوافـذ الـ Dockable اصـبـحـت ممكنة بواسطة التـعـديلات التى تمت فى Delphi 4 للـ TControl والـ TWinControl . والـ docking sites يجب أن تكون object لـ class تنحـدر من TWinControl . يجب ان يكون الـdockable control يجب أن يكون object لـ class مـشـتـقـة من الـ TControl . تنحـدر الـ TWinControl class نفـسـهـا من الـ TControl " لذلك، فـإن كـثـيـر من الـ docking site يمكن ايضـاً ان تكون نوافـذ docking . على سـبـيل المثـال، الـ ToolBar، يمكن ان يكون موقع docking site لنوافذ اخرى، أو يمكن أن يكون docking لنفسه. من الممكن ايضـاً ان يقوم بالعملية-كـ docking site لحمل نوافذ اخرى، و dockable control يمكن للمستخدمين سحبه لإلحاقة بالـ form .

تكوين الـ docking site:

يمكنك انشاء docking site لأى control بواسطة خاصية الـ DockSite . حـدد هذه الخـاصـيـة بـ True لتـسـمـح للـ control ان يقبل نافـذة اخـرى (تسـمى docked client) . الـ DockSite عـبـارة عن، CoolBar أو Panel أو Form، ولكن ان يكون أى حاوية اخرى لديها خاصية الـ DockSite .

فى اغلب الحـالات، انك تريد ايضـاً ان تحـدد خـاصـيـة الـ AutoSize لـ DockSite بـ True . عندما تكون الـ DockSite و الـ AutoSize مـحـددتان بـ True، لا يكون الـ control مرئياً فى وقت التشغيل حتى يتم وصل control آخر به. فى العـادى، هذا هو التأثير المطلوب- ان المستخدم النهائى يرى فقط الخط الخارجى لحد تمييز للـ docked control ، وليس الـ docking site الفعلى.

إنشاء dockable object:

كما ذكرنا، يمكن لـ dockable control object ان يكون أى component له خاصيتى الـ DragKind والـ DragMode . على سبيل المثال، يمكن ان يصبح الـ ToolBar عـبـارة عن dockable control . لتـفـعـل هذا، حـدد خـاصـيـة الـ DragKind للـ ToolBar بـ dkDock والـ DragMode بـ dmAutomatic .

وللتحكم فى docking control (والذى لايعد ضرورياً)، يمكنك تحديد الـ DragKind بـ dkDock (الافـتـراضى) وتـنفـيـذ OnDragOver،

OnEndDrag، و OnDragDrop . اذا كـان الـ DragMode مـحـدد بـ dmManual، استدع الـ BeginDrag لتبدأ عملية السحب.

لتـحـدد نـوع الـ object الذى يمكن ان يحتوى على dockable control عندما يتم فصله عن docking site، حدد خاصية الـ FloatingDockSiteClass الـ TMainForm، أو الى أى control آخـر. ولكن، اذا كـان الـ dockable control منحـدر من الـ TWinControl، فـهذه الخطوة غير مطلوبة لان الفـصل سيكون بصورة تلقائية.

افكار للمستخدم الخبير

- فى الحالات التى يكون فيهـا المتغيرات المتعـددة لـ class غير مذكورة فى الـ source code للبرنامج، يمكنك حـذف كل المتغيرات الا واحدة لتوفير اربع بايت لكل متغير فى الـ class objects. على سبيل المثال، افترض انك وضعت ٢٤ TSpeedButton على form، ولكنك لم تذكر أى منها بوضوح فى الـ source code للبرنامج. يمكنك حذف تعريفات الزر من الـ form class- هذا لا يحذفهم من ملف dfm. للـ form- وتوفر اربعة بايت لكل حذف. يجب ان تترك تعريفاً واحداً على الأقل من كل نوع من المتغيرات بحيث يتم ربط الـ code VCL المتعلقة بملف الـ exe code. النهائى.

- ان Modeless dialogs الموجودة على الشاشة اثناء العمليات الطويلة قد لا تتفاعل بشكل كاف لمدخلات االمستخدم. هذا يعتبر حداً من حدود الـ Windows والذى يمكنك دائماً ان يمكن أن تتعامل معه من خلال loops باستدعاء الـ ProcessMessages للـ Application العام. على سبيل المثال، اكتب time-intensive loops مثل هذه:

```
while Flag do
begin
    PerformOperation;
    Application.ProcessMessages;
    Flag := ContinueOperation;
end;
```



● اذا كنت على دراية بالـ Borland Pascal ، فقد تكون كتبت code لتنقل معلومات dialog من والى متغيرات التطبيق . فى Delphi ، قم بتعريف هذه المتغيرات ، أو توصل إليهم مباشرة من خلال الـ class public section . على سبيل المثال ، إتبع هذه الخطوات لتوضيح كيفية الوصول لـ dialog controls :

١ - أضف Button على form (module Unit1) . اختر أمر أو New SpeedButton واختر مربع الـ Standard dialog ذى الازرار على طول الحد الايمن .

٢ - أضف الـCheckBox على الـ form الثانية (module Unit2) .

٣ - أضف الـ Unit2 لتعريف الـ uses للـ Unit1 . هذا يسمح للـ event handlers للـ Button ، ولـ code اخرى فى الـ form الرئيسية ، بالوصول الى العناصر العامة للـ module الثانية .

٤ - قم بتنفيذ OnClick للـ Button فى الـmodule Unit1 كما يلى . كما توضح العبارة ، يمكنك الوصول مباشرة لخاصية الـ Checked فى الـ CheckBox1 للـ dialog . يخزن الـ dialog object قيم الـ control الخاص به ، ولذلك ، فإن آلية النقل التابعة للـ Borland Pascal لا حاجة إليها فى تطبيقات Delphi :

```
with BtnRightDlg do
if ShowModal = mrOk then
  if CheckBox1.Checked then
        ShowMessage('Checkbox is enabled')
     else
        ShowMessage('Checkbox is disabled');
```

● فى النسخ السابقة من Delphi ، تلقيت الـ GPF (خطأ الحماية العام) مباشرة عند تشغيل تطبيق اذا اشرت الى method فى form لم توجد بعد . على سبيل المثال ، فى برنامج يحتوى على dialog box ، لايمكنك استدعاء YourDialog.AnyMethod فى الـ OnCreate للـ form الرئيسية لان YourDialog لم يتم إنشاءه بعد . اذا كنت لا تزال تستخدم نسخة أولية من Delphi ، لإصلاح البرنامج ، إنقل العبارة لـ OnActivate للـ form واستخدم الـ Boolean flag لتشير اذا ما كان الـ dialog قد تم إنشاءه . هذه المشكلة لا تظهر فى Delphi 3 والنسخ الاحدث .

• كل صفحة فى الـ TabbedNotebook تعتبر TTabPage. إرجع الى هذه الـ objects من خلال خصائص الـ Objects والـ Pages. على سبيل المثال، يعرض الـ procedure التالى الـ Caption لباب الصفحة الأولى فى الـ TabbedNotebook:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  P: TTabPage;  { Reference to a TabbedNotebook page }
begin
    with TabbedNotebook1 do
       P := TTabPage(Pages.Objects[0]);  { P refers to page
      1 }
    ShowMessage(P.Caption);  { Display page tab label }
end;
```

• لتمنع المستخدمين من اختيار صفحة معينة فى الـ TabbedNotebook، قم بإنشاء الـ OnChange event، الذى له اثنين من الـ parameters بالإضافة للـ Sender : الـ NewTab، وهو يمثل الـ tab الذى يضغطه المستخدم؛ والـ AllowChange، الذى يمكنك تحديده لتشير اذا ماكانت هذه الصفحة قابلة للإختيار. على سبيل المثال، فى OnChange للـ TabbedNotebook استخدم هذه العبارة لإبطال الصفحة الثالثة (فهرس الصفحة الأولى يساوى صفراً):

```
if NewTab = 2 then
  AllowChange := False;
```

المشروعات التى يمكنك تجربتها

(١٢-١): اكتب telephone dialer dialog. يجب ان تبدو النافذة مثل هاتف الصوت. ادخل رقم هاتف من خلال الـ ShowModal اذا ضغط المستخدمون زر الـ Dial.

(١٢-٢): أضف ColorDialog وأمراً للـ SysColor من الباب السابق.

(١٢-٣): متقدم: نفذ أوامر الـ Find والـ Replace للـ TStringList.

(١٢-٤): متقدم: استخدم الـ Notebook والـ TabSet لإنشاء Options dialog متعدد الصفحات. هذا يتطلب مجهوداً زائداً عن استخدام الـ TabbedNotebook كما هو مقترح فى هذا الباب، ولكنه يعطيك مرونة اكثر. على سبيل المثال، يمكنك وضع الـ TabSet فى المكان الذى تريده، واذا كانت المسافة محدودة، يمكنك استغلال امكانية الـ TabSet على تحريك tab labels افقياً.

ملخص:

- يعتبر الـ dialog box أذن وصوت البرنامج. فى Delphi، اية form يمكن ان يكون لها صفات الـ dialog لحث المستخدمين، على عرض رسائل، وتقديم اختيارات.
- يحتوى الـ Delphi الـ Windows dialogs العامة فى الـ ColorDialog، والـ FindDialog، والـ FontDialog، والـ OpenDialog، والـ ReplaceDialog، والـ SaveDialog. يشرح ايضاً هذا الباب الـ TabSet، الـ Notebook، والـ TabbedNotebook المرتبطة، والتى تعتبر مرنة فى انشاء dialog box متعدد الصفحة.
- يمكن لـ Dialogs أن يكون modal أو modeless (تمسك به، ولكن لا تحافظ عليه). استدع الـ ShowModal method للـ form لعرض لنافذة الـ modal. واستدع الـ Show لعرض نافذة modeless dialog. لعرض Windows dialogs عامة، استدع الـ Execute methods الخاصة للـ component.
- إنك تستخدم فى المعتاد الـ TabSet، والـ Notebook معاً لإنشاء toolbar تشبه الـ VCL palette الخاصة بـ Delphi. يوضح تطبيق الـ Palette الخاص بهذا الباب هذه التقنية.
- ان الفرق الاساسى بين زوج من الـ TabSet، والـ Notebook، والـ TabbedNotebook، هو المظهر. تبدو الـ TabbedNotebooks مثل



stacked file folders ذات tabs عليها labeled بطول القمة. إن الـ TabbedNotebooks مناسب بشكل خاص فى إنشاء dialogs اختيارات متعددة الصفحة.

- إن كتابة أوامر الـ Find and Replace ليست مهمة سهلة. توفر الـ FindDialog والـ ReplaceDialog الخاصة بـ Delphi الوسليه للبحث والإبدال modeless، ان تطبيق الـ FindRepl الخاص بهذا الباب يضع الخطوات الرئيسية.
- ان برنامج العرض TextDemo الخاص بـ Delphi (انظر ملف الـ Search.pas فى دليل الـ Demos\Textdemo) يحتوى على نظام بحث للنص والذى يمكنك اخذه لعمليات الـ Find والـReplace .
- ان الـ PageControl الحديث نسبياً يبسط عملية إنشاء dialogs متعددة الصفحة، كما هو معروض بواسطة أمر الـ Options الخاص بتطبيق تعتبر كل صفحة من الـ PageControl هى TabSheet، والذى يحمل controls مثل الـ Buttons والـ CheckBoxes المرتبطة بهذه الصفحة.
- ان الـ TabControl يشبه ألـ PageControl فى انه يعرض page tabs يمكن للمستخدمين اختيارها. ولكن، يعتبر الـ TabControl شكلاً فردياً، وصفحاته ما هى الاخداع. والاستخدام الامثل للـ TabControl هو إنشاء شاشات إدخال بيانات مع مجموعات من الـ Edit التى تتغير محتوياتها إعتماداً على الـ page tab.
- استخدم خاصية الـ Constraints الجديدة لتحديد طول وعرض أى control. نفذ OnConstrainedResize للـ Form أو الـ Panel أو الـ ScrollBox.
- مع Delphi 4، يمكنك انشاء docking windows الخاصة بك. أى مثل الـ Panel له خاصية الـ DockSite يمكن ان يصبح docking site. حدد هذه الخاصية بـ True وحدد إيضاً الـ AutoSize غالباً بـ True حتى يكون الـ docking site غير مرئى فى وقت التشغيل.

- لإنشاء dockable control، حدد خاصية الـ DragKind له بـ dkDock وكذلك حدد الـ DragMode له بـ dmAutomatic . فى وقت التشغيل، يستطيع المستخدمون ان يضغوا ويسحبوا الـ control الى docking site، ويمكنهم سحبه بعيداً عن docking site لفصل الـ control .

هذا الباب يلخص فحص الجزء الثانى فى تقنيات إنشاء واجهة التطبيق الخاصة بالمستخدم باستخدام الـ Delphi components. فى الجزء الثالث، سوف ترى الجانب الآخر من عملية التطوير - المهام الداخلية التى يؤديها تطبيقك .

الجزء الثالث
التطبيق

محتويات هذا الجزء:

- **الباب الثالث عشر: تطوير تطبيقات الجرافيك.**
- **الباب الرابع عشر: تطوير تطبيقات الطابعة.**
- **الباب الخامس عشر: تطوير تطبيقات الـ MDI.**
- **الباب السادس عشر: التطوير مع الـ Clipboard، والـ DDE، والـ OLE.**
- **الباب السابع عشر: تطوير تطبيقات قاعدة البيانات.**
- **الباب الثامن عشر: تطوير الـ Charts والتقارير.**

إن تصميم واجهة تطبيق المستخدم لبرنامجك، وهى مادة هذا الكتاب حتى الآن، هى مجرد نصف لعبة كتابة برمجيات ناجحة. ويشمل النصف الثانى مميزات برنامجك - الا وهى ما يقوم به مما يجعله فريداً عن البرامج الأخرى. فى هذا الجزء، سوف تتعلم تقنيات لتطوير تطبيقات تستغل العناصر التى عرفتها من الجزء الاول والثانى.

ان اغلب التطبيقات تستخدم عناصر هامة من المواد الموضحة فى ابواب هذا الجزء. قبل قراءة الابواب القادمة، يجب ان يكون قد اعتدت نوافذ واوامر Delphi، ويجب ان تكون مستوعباً تماماً لتقنيات برمجة واجهة التطبيق الخاصة بالمستخدم وبشكل خاص الـ events وخصائص الـ component- كما هى موضع فى الجزء الثانى.

فى الابواب التالية، سوف تعرف كيفية إنشاء انواع عديدة مختلفة من التطبيقات التى تستخدم الـ components رفيعة المستوى للرسوم الجرافيكية،

الصـور المتـحـركـة، مـخـرجـات الطابعـة (MDI)، نقل بيـانات من خـلال الـ clipboard، و(DDE)، والـobject (OLE)، وActiveX، وcharts وتقارير، وإدارة قاعدة البيانات. بالرغم من ان تطبيقك فريد، بالطبع، الا انه يقع تحت أى من هذه الفئات الموضحة فى فصول هذا الباب.

الباب الثالث عشر
تطوير تطبيقات الجرافيكية

محتويات هذا الباب:

- Components
- معلومات حول canvas
- الرسم والتلوين
- تقنيات برمجة الرسوم الجرافيكية
- Picture dialogs.
- Animations

بغض النظر عن مهاراتك الفنيــــة، مع مســـاعدة من خاصية الـ Canvas لـ Delphi، يمكنك رسم وتلوين أى شكل تتخيله. بالإضـافـة الى هذا، فـإن الجرافيك وانواع البيانات و الـ procedures والـ functions الخاصة بـ Delphi قد بسطت الى حد كبير مهام البرمجة مثل العمل بالـ bitmaps، ووقراءة وكتابة الملفات ذات الاحجام الكبيرة.

فى هذا الباب، سوف اقدم تقنيات الجرافيك الخاصة بـ Delphi وسأوضح كيفية استخدام خاصية الـ Canvas المكثفة، والتى توفرها كثير من الـ components. سوف اغطى ايضاً الموضوعات الجرافيكية المتعلقة مثل الصور المتحركة، التعامل مع ملف الجرافيك، (offscreen bitmaps)، dialogs pictune، و arag-and-drop objects.

Components :

فيما يلى قائمة بـ components الجرافيك الخاصة بـ Delphi :

- **Animate**: استخدم هذا الـ component لإضافة نوافذ صـور متحـركة للـ form أو الـ dialog . يمكنك ان تشغل ملفات الـ avi. (السـمعيـة البـصرية) المعيارية، كما يمكنك الاختيار من الصور المتحركة المعيارية للـ Windows كتلك التى تراها عند نسخ وحذف ملفات باستخدام الـ Explorer . الـ Palette : Win32.

- **Image**: استخدم هذا الـ component لإدخال bitmaps ، ايقـونات ، وملفات كبيرة (أو metafiles) فى form. ان الـ Image object يعد غلافاً يحتوى على object جرافيكى فى خاصية الـ Picture التابعة له . الـ Palette : Additional.

- **MediaPlayer**: استخدم هذا الـ component للوسائط المتعددة، وهو كامل الميزات لإقامة برمجيات صوتية وجرافيكية متحركة . كما يوضح هذا الباب ، بقليل من ضربات للوحة المفاتيح ، يمكنك إنشاء viewer ملف الـ avi video. (المرئية المسموعة) الـ Palette : System .

- **OpenPictureDialog،SavePictureDialog**: يوفر هذان الـ dialogs components معيارية لفتح وحفظ ملفات جرافيكية . الـ Palette : Dialogs.

- **PaintBox**: استخدم هذا الـ component لإضافة إمكانيات الرسم والتلوين للـ components مثل الـ Panel التى ليس لها خاصية Canvas. يمكنك ايضاً استخدام الـ PaintBox لتحديد الرسم على مساحة مربعة واحدة أو أكثر فى نافذة الـ form . الـ Palette : System .

- **Shape**: استخدم هذا الـ component لإنشاء object هندسية الشكل مثل المستطيلات وأشكال بيضاوية ممتلئة وغير متملئة . اختر الـ object الذى تريده بتحديد خاصية الـ Shape للـ Shape (الخاصية و الـ component لهما نفس الاسم) . الـ Palette : Additional .



معلومات حول الـ Canvas:

ان كل العمليات الجرافيكية تحدث فى اطار خاصية الـ Canvas، التى يوفرها العديد من الـ components . على سبيل المثال، ان للـ form خاصية Canvas، وكذلك الـ TBitmap، والـ TComboBox، والـ TListBox، و الـ TPaintBox وغيرها. وتمثل خاصية الـ Canvas الـ Windows Graphics Device Interface (GDI) ، وهى توفر سطح مستقل لرسم objects جرافيكية على النوافذ. (يغطى الباب التالى طبع الجرافيك والنص). من الجيد ان ترى الـ Canvas على انها سطح يلون عليه برنامجك النص والرسوم الجرافيكية- إنها ليست "سطحاً" ، أنما أكثر منها قناة تنساب من خلالها اوامر الجرافيك الى الشاشة، حيث تصبح بصرية.

ملحوظة: الى أولئك المعتادون على برمجة الـ Windows، يجب أن الـ Canvas ماهو إلا object-oriented encapsulation للتعامل مع HDC. لإنشاء رسوم جرافيكية، تقوم الـ methods الموجودة فى الـ TCanvas class باستدعاء الـ functions الـ Windows GDI .

الرسم مع الـ Canvas:

تعتبر خاصية الـ Canvas شاملة للـ Windows GDI . ودائماً الـ Canvas، تعتبر جاهزة للاستخدام. هى object بكل معناه، وتوفر الـ TCanvas class عشرات الخصائص، methods والـ events . بشكل عام، إنك تستخدم الـ Canvas بطريقتين:

- لتشكيل مخرجات جرافيكية بتعيين قيم ألوان، أنماط، fonts، وقيم أخرى لخصائص الـ Canvas.
- لإنتاج objects بصرية باستدعاء methods جرافيكية، بعضها أيضاً يوفر خدمات معاونة مثل تحديد عرض للشاشة بالـ pixel لـ string .

ملحوظة: أنظر البـــاب التالى لمزيد من المعلومات حول عرض وطباعة نص مع canvas text output method.

إن خـاصـيـة الـ Canvas غـيـر مـتـاحـة فى وقت التـصـمـيم فى نافـذة الـ Object Inspector . لاسـتـخـدام الـ Canvas ، يجب أن تكتب عـبـارات تحـدد قـيم لخـصـائص الـ Canvas وتستدعى الـ Canvas methods . على سبيل المثال ، لرسم مستطيل أزرق ممتلئ بخطوط قطرية صفراء فى نافذة الـ form ، أدخل هذه العبارات فى OnPaint للـ form :

```
with Canvas do
begin
    Pen.Color := clBlue;
    Brush.Color := clYellow;
    Brush.Style := bsDiagCross;
    Rectangle(10, 10, 100, 100);
end;
```

لإدخـال هذا الـ code ، إبدأ تطبيـقاً جـديداً ، اختـر Form1 فى الـ Object Inspector ، اضـغـط الـ Events page tab لهـذه النافـذة ، واضـغـط مـرتين فى حـقل القيـمة للـ OnPaint . أضـف السطور السـابقة للـ event handler النـاتج الذى ينشـئه Delphi ، ثم اضغط F9 لتشغيل البرنامج .

وهذه الجزئية من البرمجة تستخدم عبارة with التى تخبر الـ Delphi أن يستخدم الخـصـائص و الـ methods فى الـ Canvas . ولأن هذه العـبـارة تظهـر فى TForm1 method ، فـإن الـ Canvas الخـاص بالـ form هو الذى يتم ذكـره ، ولذلك ، تظهـر المخرجات فى نافذة البرنامج الرئيسية فى هذه الحالة . تقوم العبارات الثلاث الأولى بتعيين قيم لإثنين من خصائص الـ Canvas : الـ Pen و الـ Brush ، مما يحدد الخصائص الفرعية لهـذه العناصر كـما هو موضـح . تقوم العـبارة الأخيرة باستدعـاء الـ Canvas method ، وهو الـ Rectangle لرسم object جـرافيكى باسـتـخـدام القـيم المعـينة . لإنشاء أغلب أنواع الجرافيك ، فإنك تستخدم خطوات مشابهة :

- تعيين قيم لخصائص والخصائص الفرعية للـ Canvas .
- استدعاء الـ Canvas methods لإنتاج مخرجات بصرية .

الرسم بالـ Shapes:

يقـدم Delphi طريقـة أخـرى لإنشـاء الـ Shapes . فـبـدلاً من اسـتـدعـاء الـ Canvas methods لرسم object فى وقت التشغيل ، يمكنك إضافة shape على



الـ form . إن النتيجة النهائية من الجانب البصرى واحدة ، ولكن الـ Shape يعطيك سبل ترتيب الـ objects الجرافيكية فى وقت التصميم بالضغط والسحب وبتعيين قيم خاصة . على سبيل المثال ، أضف من الـ Additional palette على الـ from من الفصل السابق ، أو استخدم from جديدة . اختر قيمة خاصية Shape مثل الـ stRoundRect ، أو أدخل هذه العبارة فى الـ OnCreate أو أى event handler آخر للـ form .

```
Shape1.Shape := stRoundRect;
```

لتغيير لون ونمط الـ Shape فى وقت التصميم ، استخدم الـ Object Inspector لتعيين قيم للخصائص الفرعية للـ Pen والـ Brush . تؤثر الـ Brush على داخل الـ object . وتؤثر الـ Pen على حد تمييز الـ object . جرب مواصفات مختلفة ، أو عين قيم فى وقت التشغيل بعبارات مثل التالية فى OnCreate أو فى event handler آخر للـ form :

```
with Shape1 do
begin
  Shape := stRoundRect;
    Brush.Color := clLime;
    Brush.Style := bsCross;
    Pen.Color := clNavy;
    Pen.Width := 3;
end;
```

لأن كثير من الـ components لها خاصية Canvas ، اذا لم تظهر صور الجرافيك كما هى متوقعة ، قد يكون السبب عبارة with التى تشير الى Canvas خاصة بالـ object وأنت تريد أن تستخدم خواص الـ form . أيضاً لا تستخدمها مع عبارة with ، أو تستخدم عبارات مثل MainForm.Canvas .

خصائص الـ Canvas:

إن خاصية الـ Canvas توفر ثمان خصائص فرعية لتشكيل الجرافيك . يمكنك تعيين قيم الى هذه الخصائص فى وقت التشغيل فقط ، عادة فى الـ OnPaint event handler أو OnPaint للـ form . وفيما يلى وصف مختصر لكل خاصية :

- **Brush:** توفر اللون والنمط داخل حدود الدائرة أو المستطيل أو الشكل الخماسى، وكذلك لون الخلفية للنص. ليس لها تأثير على السطور أو لون واجهة النص. قم بتعيين قيم للخصائص الفرعية لل Style و Brush Color .

- **ClipRect:** يضع الصور الجرافيكية داخل حدود هذا المستطيل. إنه مساو عادة فى الحجم لل client area بالنافذة. يمكنك تعيين قيم جديدة للقيم الفرعية وهى الـ Left والـ Top والـ Right والـ Bottom Integer الخاصة به لوضع المخرجات الجرافيكية فى منطقة أخرى. (ملحوظة: أنظر الـ PaintBox component للتعامل مع method أسهل).

- **CopyMode:** يحدد كيفية تجميع البت عند استدعاء الـ CopyRect method لل Canvas، والذى تستطيع استخدامه لنسخ خاصية الـ Canvas لإحدى الـ object الى object أخر. على سبيل المثال، حدد الـ CopyMode بـ cmNotSourceCopy لعكس البيكسلات قبل نسخها الى Canvas آخر.

- **Font:** تعيين الخصائص الفرعية لإختيار أنماط الـ font للنص الذى ترسمه باستدعاء الـ TextRect methods والـ TextOut لل Canvas. الـ Font هذا لا علاقة له بخاصية الـ Font التابعة لل form- حدد خصائص الـ Canvas.Font قبل رسم النص.

- **Handle:** من الـ GDI functions التى تتطلب handle، إلى HDC، قم بتمرير Canvas.Handle لل HDC parameter. هذا يجعل من الممكن استدعاء الـ GDI functions التى لا يحتويها الـ Canvas.

- **Pen:** تؤثر على السطور والـ outlines، إنك تعين قيم للخصائص الفرعية والتى هى: الـ Color، والـ Style، والـ Mode، والـ Width لل Pen.

- **PenPos:** يعطى موضع الـ pen الداخلى (PenPos.X, PenPos.Y) الذى يحدد أين ستظهر المخرجات الجرافيكية التالية. بالرغم من أن Delphi يسمح بتعيين قيم جديدة لل PenPos، يجب عليك بدلاً من ذلك استدعاء الـ MoveTo method الخاص بالـ Canvas لتغيير الموضع الداخلى لل Pen.

● **Pixels:** توفر two-dimensional array للـ pixels المنفردة التابعة للـ Canvas . يمثل كل عنصر من الـ array قيمة الـ TColor . على سبيل المثال ، التعبير Pixels[0,0] (استخدم أقواساً مربعة ، يدخل لون الـ pixel فى الإحداثية (0,0) . اذا أردت أن تصنع الجرافيك بالطريقة الصعبة) يمكنك أيضاً تلوين الـ Pixels الفردية بتعيين قيم لون جديدة لهذا الـ array .

الـ events و methods الـ Canvas:

توفر خاصية الـ Canvas عدد من الـ methods يمكن أن يساعدك على ترويض الجنون الذى يمر به كل مبرمج ، آجلاً أو عاجلاً ، مع الـ Windows GDI . إن العديد من الـ shape-Producing methods مثل الـ Arc ، Ellipse ، FloodFill ، Polygon ، Rectangle و RoundRect توازى مباشرة الـ GDI functions التى تحمل نفس الاسماء . بعد أن تعتاد على التقنية العامة لاستدعاء الـ Canvas methods ، فستكون سهلة الاستخدام . على سبيل المثال ، لرسم شكل بيضاوى ، أدخل عبارات مثل هذه فى : الـ OnPaint

```
Canvas.Pen.Color := clMaroon;
Canvas.Brush.Color := clYellow;
Canvas.Ellipse(10, 20, 100, 100);
```

قم بتعيين قيم لخصائص الـ Canvas قبل كل استخدام خصائص الـ Canvas فى OnPaint . على سبيل المثال ، لرسم خط أزرق ، عين clBlue للخاصية الفرعية للـ Canvas Pen.Color ، ثم استدع الـ MoveTo methods والـ LineTo . لا يمكنك أن تقوم بالتشكيل المسبق لخصائص الـ Canvas فى OnCreate للـ form لأن الـ OnPaint يتلقى device context من الـ Windows ، وهذه الـ devicecontext متوفرة لبرنامجك عن طريق الـ Canvas . لذا ، فالـ Canvas فى الـ OnPaint يحتوى أى قيم لخاصية معينة خارج الـ event handler .

الـ methods الأقل وضوحاً لخاصية الـ Canvas والتى يمكنك استدعاؤها تتضمن ما يلى :

● **CopyRect:** ينسخ كل واحدة من خصائص الـ Canvas أو جزء منها الى أخرى . حدد خاصية الـ CopyMode لتحدد كم pixels تم تجميعه فى النتيجة

النهائية. أنظر الـ CopyMode فى الـ Delphi online help لمعرفة المواصفات التى يمكنك استخدامها.

● **Draw:** يرسم object التابع لـ TGraphic class، والتى تعد السلف المباشر للـ TIcon classes والـ TBitmap والـ TMetafile. يمكنك تمرير أى أيقـونة أو bitmap أو ملف كبير للـ Draw.

● **DrawFocusRect:** يرسم مستطيل باستخدام عبارات الـ OR المنطقية. استدع وقتاً آخر الـ arguments المطابقة لمسح المستطيل استخدم هذا الـ method لرسم خطوط سحب وإسقاط خارجية.

● **FrameRect:** يرسم مستطيل غير ممتلئ باستخدام الـ Pen الحالية، ولكن متجاهلاً فرشاة الـ Canvas استخدم الـ FrameRect بدلاً من الـ Rectangle عندما تريد outline عادى دون أن تحدد خصائص الـ Brush.

● **StretchDraw:** هذه هى نفسها الـ Draw، ولكنها تمد أو تقلص الأيقونة أو bitmap أو الملف الكبير ليتلائم داخل المستطيل المحدد.

خاصية الـ Canvas تعرف اثنين من الـ events التى قد تكون مفيدة فى احوال خاصة. فى اغلب التطبيقات، لن تحتاج الى توفير events لهما- فقط كن على دراية بهما اذا، ما احتجت الى خدماتهما الخاصة. و الـ event handlers هما:

● **OnChange:** يتم استدعاؤه بعد ان تتغير قيمة الـ Canvas .

● **OnChanging:** يتم استدعاؤه قبل تتغير خاصية الـ Canvas مباشرة.

لان خاصية الـ Canvas غير متاحة فى وقت التصميم، لا يمكنك استخدام الـ Object Inspector لإنشاء handler لهذين الـ event. لاستخدامها، قم بتعريف الـ procedures فى الـ form class :

```
TForm1 = class(TForm)
  procedure MyOnChange(Sender: TObject);
     procedure MyOnChanging(Sender: TObject);
...
end;
```

قم بتعيين اسماء الـ procedure للـ OnChange والـ OnChanging في OnCreate الخاص بالـ form :

```
Canvas.OnChange := MyOnChange;
Canvas.OnChanging := MyOnChanging;
```

قم بتنفيذ الـ procedure لتنفيذ اية اعمال تحتاجها . على سبيل المثال ، اكتب الـ OnChange كمايلى :

```
procedure TForm1.MyOnChange(Sender: TObject);
begin
  SavedPenColor := Canvas.Pen.Color;
end;
```

Caution

تحذير: يقوم الـ Project Manager الخاص بـ Delphi بحذف التعريفات السابقة اذا كان الـ event handler لا تحتوى على عبارات أو تعليقات عند إنشاء event handler متعددة ، لا تقم بعملية الـ compile للبرنامج الا بعد إدخال عبارة أو تعليقاً واحداً على الأقل فى كل handler procedure .

الرسم والتصوير:

تقوم تطبيقات الـ Windows برسم shapes منفردة مثل الخطوط والمستطيلات الممتلئة . وهى تصور الـ objects الجرافيكية لتحافظ على خداع النوافذ الواقعة فوق بعضها على الحاسب المكتبى . قد يكون على البرنامج ان يرسم الـ object مرة واحدة فقط ، ولكنه يجب ان يكون مستعداً دائماً ان يصوره اذا قام المستخدم ، مثلاً ، بإخفاء ثم إظهار نافذة التطبيق .

ويظهر الاختيار البسيط الفرق بين الرسم والتصوير . أضف الـ Button object على الـ form . إنشئ OnClick للـ Button مع البرمجة الموجودة فى القائمة (١٣-١) . قم بتشغيل البرنامج واضغط الزر لتحدد الـ outline للزر بمستطيل احمر . قم باخفاء واظهار النافذة ، ويختفى المستطيل .

القائمة (١٣-١): استخدام OnClick لعرض الرسم والتصوير

```
procedure TForm1.Button1Click(Sender: TObject);
```

```
var
  X1, Y1, X2, Y2: Integer;
begin
    Canvas.Pen.Color := clRed;
    Canvas.Pen.Width := 2;
    with Button1 do
    begin
     X1 := Left - 3; Y1 := Top - 3;
       X2 := Left + Width + 4; Y2 := Top + Height + 4;
       Canvas.RoundRect(X1, Y1, X2, Y2, 4, 4);
     end;
end;
```

اترك البـرنامج الاختبـارى وارجع الى Delphi. ادخل OnPaint للـ form وانسخ العبـارات وتعـريف الـ var فى هذا الـ procedure. احـذف OnPaint. والآن عندما تقوم بتشغيل البرنامج، فإنه يعطى outline للـ Button من البداية، ولا يختفى الـ outline عندما تقوم باخفاء وإظهار النافذة.

من الاسهل والأفضل عادة ان تحذف الـevent handler بإزالة التعريفات والعبـارات الخـاصـة به، تاركـاً الـ begin والـ end فقط. لا تحـذف الـ procedure بأكمله، هذا قـد يحير مولد الـ Delphi code. على سبيل المثال، لإزالة برنامج OnClick الخـاص بالـ Button، قم بتـقليل الـ procedure الموجـود فى القـائمـة (١٣-١) الى مايلى:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
end;
```

عندمـا تقوم بعـد ذلك بإجـراء عـمليـة الـ compile للف الـ source code، يحـذف الـ Project Manager الذكى لـ Delphi هيكل وتعـريف الـ procedure مع أى إشارات الى هذا الـ procedure فى الـ object events.

فى اغلب الحـالات، يمكنك إضـافـة عـبـارات الرسم على الـ form أو فى OnPaint للـ PaintBox. ولكن اذا رسـمت objects فى procedures اخـرى، يجب ان توفـر code ايضـاً لإعـادة إنشاء تلك الـ objects فى الـ OnPaint. اذا لم

تفعل هذا، فإن (رسوم الجرافيك) الخاصة بك تختفى عندما يخفى المستخدمون نافذة التطبيق ويظهرونها.

قم باستدعاء الـ Invalidate method للـ form لإخبار الـ Windows . ان الـ form (أو الـ control) تحتاج الى إعادة تصوير . قم باستدعاء الـ Update method للـ form لإعادة التصوير فوراً . على سبيل المثال، إفترض أن برنامجك له OnPaint event handler يرسم بعض الـ objects الجرافيكية المؤسسة على متغيرات عامة . اذا تغيرت هذه المتغيرات، لا ترسم الـ object الجديد- بدلاً من ذلك، أدخل هذه العبارات لإعادة تصوير الجرافيك بإجبار الـ OnPaint على الحدوث:

```
MainForm.Invalidate;
MainForm.Update;    { optional }
```

لا تفعل هذا أبداً داخل OnPaint . إن هدف هذه العبارات هو أن تجعل الـ OnPaint يحدث- وتنفيذها داخل الـ OnPaint تجعل البرنامج يعيد تصوير نفسه الى مالا نهاية . ويخبر الـ Invalidate الـ Windows إن الـ object يحتاج الى تصوير . يصدر الـ Windows رسالة wm_Paint للـ object عندما لا يكون هناك رسائل أخرى . والـ Update ، والذى يعتبر اختيارياً، يخبر الـ Windows أن يصدر رسالة الـ wm_Paint على الفور.

الـ PaintBox component:

أضف الـ PaintBox على form أو حاوية أخرى مثل الـ Panel لمنحه إمكانات الجرافيك، إن كل الأحداثيات متصلة بالحاوية. على سبيل المثال، أضف Panel على الـ from وقم بمدها قليلاً لتفسح مكاناً للـ PaintBox. اختر الـ PaintBox من لوحة الـ System واضغط مؤشر الفأرة داخل الـ Panel . لإعادة وضع الـ object، ببساطة اسحب الـ Panel- لا يجب عليك أن تختار الـ PaintBox أيضاً.

لقد أعطيت الـ Panel لتوك إمكانات جرافيك جديدة . يمكنك استخدام هذه التقنية لرسم objects، وbitmaps، ورسوم جرافيكية أخرى فى الـtoolbar أو الـstatus ilne . على سبيل المثال، لرسم مربع أزرق داخل الـ Panel ، أدخل

العبارات التالية فى OnPaint الخاص بالـ PaintBox1 . (تأكد من إنشاء الـ event handler للـ PaintBox، وليس للـ form) . ها هو الـ procedure كاملاً:

```
procedure TForm1.PaintBox1Paint(Sender: TObject);
begin
  with PaintBox1.Canvas do
    begin
      Pen.Color := clNavy;
        Rectangle(0, 0, 100, 100);
      end;
end;
```

Note

ملحوظة: لاحظ أن عبارة الـ with تشير الى الـ PaintBox1 Canvas . اذا كانت هذه العبارة تشير إلى الـ Canvas وحدها ، فإنها ستشير الى الـ Canvas الخاص بالـ form بدلاً من ذلك الخاص بالـ PaintBox. كن متنبهاً دائماً الى أن كل الـ components البصرية لها خصائص Canvas وكن متأكداً من أنك تشير الى الصحيح منها حتى تذهب مخرجات الجرافيك الخاص بك الى المكان الذى تريده .

يمكنك أيضاً استخدام الـ PaintBox لقصر الرسم على مستطيل بعينه فى form أو حاوية أخرى. لتقسيم الـform أو حاوية أخرى الى قطاعات متعددة، يمكنك إدخال أى عدد من الـ PaintBoxes .

استخدام الـ Pens والـ Brushes:

إن كل خاصية Canvas لها خصائص الـ Pen والـ Brush . والـ Pen تحدد لون ونمط الخطوط، وكذلك outlines للـ objects مثل المستطيلات والأشكال البيضاوية. وتحدد الـ Brush أنماط وألوان داخل الـ objects . وللـ Pens أربعة خصائص هامة:

- **Color:** تؤثر على الخط و outline، ولكن ليس على ألوان داخل الـ object . عين أى قيمة TColor أو ثابت مثل الـ clRed أو الـ clLime للـ Pen.Color . لتتماشى مع ألوان شاشة الحاسب المكتبى، عين ثابتاً مثل الـ clWindow أو الـ clWindowText .

• **Mode:** تحدد method المنطقى لرسم الخطوط و outlines . عين ثابتاً مثل الـ pmBlack (اسـود دائمـاً) أو الـ pmNotCopy (عكس لون الـ Pen) للـ Pen.Mode . استخدم الـ pmXor بحيث يمكنك إعادة رسم خط لمسحه وتخزين خلفيته الأولى .

• **Style:** اختر نمط خط منقط أو dashed ، أو solid . على سبيل المثال ، عين ثابتـاً مـثل psSolid أو psDashDot للـ Pen.Style فى الـ Windows ذى الـ 16-bit والـ Windows 95 ، فـإن الخطوط التى يزيد عـرضـهـا عن pixel واحـد تكون solid دائماً . والـ Windows 95 والـ Windows NT يمكناك من تحديد نمط الخطوط السميكة والرفيعة .

• **Width:** يحـدد العـرض بالـ pixels للخطوط و outlines . فى الـ Windows ذى الـ 16-bit ، يجب أن تكون هذه الخـاصـيـة مـسـاوية لواحـد (الإفتراضى) لكى تستخدم Pen Style آخر غير الـ psSolid .

للحصول على قائمة كاملة لثوابت الـ Color والـ Mode والـ Style ، أنظر هذه الخصائص الخاصة بالـ TPen class فى الـ online help الخاصة بـ Delphi . أو ، اذا كـان لديك library source code وقت التـشـغيل الخـاصـة بـ Delphi ، تصفح ملف الـ Graphics.pas .

وللـ Brushes ثلاث خصائص هامة :

• **Bitmap:** يعين TBitmap object بـبـعـد يصل الى ٨ × ٨ pixels . تنسخ الفرشاة الـ Bitmap لتملأ الـ objects الداخلية .

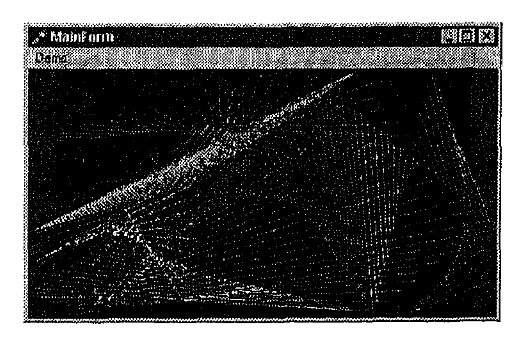
• **Color:** هذا هو نفسه كما فى الـ Pen .

• **Style:** تحدد نمط للاستخدام لتصوير الـ objects الداخلية .

على سـبـيل المثـال ، عين ثابتـاً مـثل الـ bsCross أو الـ bsFDiagonal للـ Brush.Style . استـخدم الـ bsSolid عند تعيين الـ Bitmap . لرسم outline مع الخلفية ظاهرة من خلاله ، حدد الـ Style بـ bsClear . للحصول على قائمة كاملة بثوابت الـ Style ، أنظر خاصـيـة الـ Style للـ TBrush class فى الـ online help الخاصة بـ Delphi .

إن الـ Pens والـ Brushes لهـا methods مـثل الـ Create والـ Free والتى تعـتبر مـشتركة بين كـثيـر من الـ components للـ Brushes والـ Pens أيضـاً والـ OnChange والذى يمكنك استـخدامه لتخبر البرنامج بأية تغييرات لخصائص الـ Pen والـ Brush ، بالرغم من أن استخدام هذا الـ event نادراً ما يكون ضرورياً .

يعـتـبر تطبيق الـ PolyFlow على القـرص المدمج واحـداً من أقـدم البـرامج الإختـبارية المعـيارية للـ Pascal توجد الملفـات فى دليل الـ Source\PolyFlow . يعد الـ PolyFlow مثالاً جيداً على الرسم مقابل التصوير ؛ وإنه يوضح أيضاً كيفية استـخدام Timer object لإنشاء صور جرافيكية متحركة . يوضح شكل (١٣-١) عرض الـ Polyflow . القائمة (١٣-٢) تعطى الـ source code للبرنامج .



شكل (١٣-١): الـ Polyflow، والذى يعد برنامج معيارى قديم مع واجهة تطبيق Delphi الحديثة، يوضح الرسم مقابل التلوين، ويوضح كيفية استخدام Timer object لإنشاء صور جرافيكية متحركة

القائمة (١٣-٢): PolyFlow\Main.pas

```
unit Main;

interface

uses
  SysUtils, Windows, Messages, Classes, Graphics,
    Controls, Forms, Dialogs, Menus, ExtCtrls;

const
    maxIndex = 100;     { Maximum number of lines visible }
```



```
    dx1: Integer = 4;   { "Delta" values for controlling }
     dy1: Integer = 10;  {  the animation's personality.  }
     dx2: Integer = 3;
     dy2: Integer = 9;

  type
   LineRec = record     { Line ends and color }
      X1, Y1, X2, Y2 : Integer;
        Color: TColor;
       end;

  type
   TMainForm = class(TForm)
      MainMenu1: TMainMenu;
        Demo1: TMenuItem;
        Exit1: TMenuItem;
      (continued)
      Listing 13-2 (continued)
        Timer1: TTimer;
        procedure Exit1Click(Sender: TObject);
        procedure FormCreate(Sender: TObject);
        procedure Timer1Timer(Sender: TObject);
        procedure FormPaint(Sender: TObject);
        procedure FormResize(Sender: TObject);
     protected
      procedure CreateParams(var Params: TCreateParams);
          override;
          private
      LineArray: array[0 .. maxIndex _ 1] of LineRec;
        Index: Integer;     { Index for LineArray }
        Erasing: Boolean;  { True if erasing old lines }
        function Sign(N: Integer): Integer;
        procedure InitLineArray;
        procedure MakeNewLine(R: TRect; Index: Integer);
        procedure DrawLine(Index: Integer);
       public
```

```
     { Public declarations }
    end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}

procedure TMainForm.CreateParams(var Params: TCreateParams);
begin
  inherited CreateParams(params);
     with Params.WindowClass do
    {   Repaint the window automatically when resized   }
    Style := Style or cs_HRedraw or cs_VRedraw;
end;

{ Return _1 if n < 0 or +1 if n >= 0 }
function TMainForm.Sign(N: Integer): Integer;
begin
  if N < 0 then Sign :=  _1 else Sign := 1;
end;

{ Erase LineArray and set X1 to -1 as "no line" flag }
procedure TMainForm.InitLineArray;
var
  I: Integer;
begin
  Index := 0;
     Erasing := False;
     FillChar(LineArray, SizeOf(LineArray), 0);
     for I := 0 to maxIndex _ 1 do
       LineArray[I].X1 := _ 1;
end;
```

```
{ Create new line, direction, and color }
procedure TMainForm.MakeNewLine(R: TRect; Index: Integer);
 procedure NewCoord(var C, Change: Integer; Max: Integer;
     var Color: TColor);
      var
   Temp: Integer;
     begin
      Temp := C + Change;
      if (Temp < 0) or (Temp > Max) then
      begin
       Change := Sign(-Change) * (3 + Random(12));
          repeat
           Color := RGB(Random(256), Random(256), Random(256));
              Color := GetNearestColor(Canvas.Handle,
         Color)
             until Color <> GetBkColor(Canvas.Handle);
         end else
       C := Temp;
       end;
begin
 with LineArray[Index] do
     begin
      NewCoord(X1, dx1, R.Right, Color);
         NewCoord(Y1, dy1, R.Bottom, Color);
         NewCoord(X2, dx2, R.Right, Color);
         NewCoord(Y2, dy2, R.Bottom, Color)
       end
end;

{ Draw or erase a line identified by Index }
procedure TMainForm.DrawLine(Index: Integer);
begin
 with Canvas, LineArray[Index] do
     begin
      Pen.Color := Color;
         MoveTo(X1, Y1);
```

```
      LineTo(X2, Y2);
       end;
end;

{ Draw some lines at each timer interval }
procedure TMainForm.Timer1Timer(Sender: TObject);
var
    R: TRect;
    I, OldIndex: Integer;
begin
  R := GetClientRect;
    for I := 1 to 10 do  { 10 = number of lines }
    begin
      OldIndex := Index;
        Inc(Index);
        if Index = maxIndex _ 1 then
        begin
          Index := 0;       { Wrap Index around to start }
             Erasing := True;  { True until window size
        changes }
            end;
      if Erasing then
           DrawLine(Index);  { Erase old line }
             LineArray[Index] := LineArray[OldIndex];
         MakeNewLine(R, Index);
         DrawLine(Index);    { Draw new line }
       end;
end;

{ Paint or repaint screen using data in LineArray }
procedure TMainForm.FormPaint(Sender: TObject);
var
  I: Integer;
begin
  with Canvas do
      for I := 0 to maxIndex _ 1 do
```

```
        if LineArray[I].X1 >= 0 then  { Draw non-flagged lines }
            DrawLine(I);
end;

{ Start new lines when window size changes }
procedure TMainForm.FormResize(Sender: TObject);
begin
 InitLineArray;   { Erase LineArray and reset globals }
end;

{ Initialize globals and LineArray }
procedure TMainForm.FormCreate(Sender: TObject);
begin
 with Canvas.Pen do
   begin
    Style := psSolid;
      Width := 1;
      Mode := pmXor;
     end;
     Randomize;
     InitLineArray;
  end;

{ End program }
procedure TMainForm.Exit1Click(Sender: TObject);
begin
 Close;
end;

end.
```

إن OnTimer الخـــاص بالـ PolyFlow (أنظر الـ Timer1Timer) يرسـم عشرة خطوط من الصورة المتحركة. ويمكن أن يرسم خطاً واحداً فقط، ولكن هذا سيؤدى الى إبطاء التأثير بشكل غير مقبول. ويقوم OnPaint الخاص بالـ form بإعادة رسم النافذة بأكملها بحيث، عندما تريد إخفاءها ثم إظهارها مرة أخرى،

يخرن البرنامج الحالة الحالية للصورة المتحركة . وهذه تقنية Windows جرافيكية نموذجية- تقوم أجزاء من البرنامج برسم الأشكال وتخزين الـ parameters الخاصة بها (أنظر الـ LineArray فى القائمة). للحفاظ على النافذة، يستخدم OnPaint تلك الـ parameters لإعادة تصوير الجرافيك الخاص بالبرنامج .

حاول تشغيل الـ PolyFlow وقم بتغطية جزء من نافذته. لاحظ أن الـ Timer events تستمر فى التشغيل . بالإضافة الى إنشاء صور متحركة، يمكنك استخدام نفس هذه التقنية لبرمجة عمليات خلفية أخرى، ولكن إنتبه ألا تغلق النظام بتنفيذ عمليات طويلة فى الـ Timer event .

تقنيات برمجة الجرافيك:

إن كل تطبيقات Delphi الجرافيكية والمؤسسة على الـ form تستخدم وحدة الـ Graphics، والتى توفر classes مثل الـ TPen، TBrush، TFont، TCanvas، وتعريفات متعلقة بها. وهذا الفصل يغطى بعض الـ classes الأخرى والتقنيات الجرافيكية المتقدمة مثل الـbitmaps والملفات الجرافيكية وكيفية إنشاء objects السحب والإسقاط .

علاقات class الجرافيك:

عندما تبدأ فى تصميم برنامج جرافيكى، قد تكون غير واثقاً أن تستخدم components أو Shape أو object من الـ TPicture class. لتقر أفضل مسار لبرنامجك، إفهم أولاً العلاقات الموجودة بين الـ classes التالية :

- **TCanvas:** كما ذكرنا، هذا الـ class توفر سطح يمكنك أن ترسم وتصور عليه الجرافيك باستدعاء الـmethods وتعيين قيم الخصائص. إنك لا تنشئ أبداً TCanvas object مستقل- أنت تستخدم هذا الـ class دائماً كخاصية الـ Canvas لـ object آخر مثل الـ PaintBox أو الـ form .

- **TGraphic:** هذا الـ class هو السلف المباشر للـ TBitmap classes والـ TIcon والـ TMetafile. لا تنشئ أبداً TGraphic object مستقل. بدلاً من ذلك، يمكنك تعيين bitmap object أو أيقونة أو ملف كبير لأى TGraphic parameter أو متغير أو خاصية. وتعمل للـ TGraphic أيضاً كـ class مجردة لكل

أنواع بيـانات الصـور فى Delphi، بما فى ذلك لـ add-in classes مـثل الـ JPEG (من وحدة الـ JPEG) زائد format ملفات آخرى .

● **TPicture:** هذا الـ class تنشىء حـاوية تحـمل الـ TGraphic object، والذى يمكن أن يكون bitmap أو أيقونة أو ملف كبير . إن هذهmethods تسمح لك بالتعامل مع الملف الخاصة بالـ TPicture وتحدد نوع صورة من إمتداد اسم الملف الخـاص بهـا، والذى يبسط عـملية التـعامل مع ملف الجرافيك . سـوف تجد الـ TPicture class مـسـتخـدمة كـخاصيـة الـ Picture فى objects أخـرى مـثل الـ Image component، يمكنك أيضاً إنشاء TPicture object مستقل - على سبيل المثال، لتوفر حاوية bitmap تم تحميلها من ملف قرص .

● **TGraphicsObject:** هذا الـ class هى السلف المبـاشـر لـ TFont classes والـ TPen والـ TBrush. استخدم الـ TGraphicsObject على إنها نوع function parameter أو procedure بحـيث يمكنك تمرير Fonts أو pens أو brushes كـ arguments للنظم الفرعية . لا تنشىء أبداً object مستقل من class لـ TGraphicsObject .

● **TGraphicControl:** تعـتبـر الـ component class هذه هى السلف لـ component classes الجرافيكية مثل الـ TSpeedButton، والـ TTabButton (المسـتـخـدمـة بواسطة الـ TabbedNotebook)، TShape، TPaintBox، TImage، والـ TCustomLabel (السلف المبـاشـر لـ TLabel). تـوفـر الـ TGraphicControl خاصية Canvas، والتى، على سبيل المثال، يستخدمها الـ SpeedButton لعـرض glyph bitmap الخـاص به . وغـالبـاً لن تحـتـاج الـ TGraphicControl فى تطوير التطبيق . إنهـا مـخـصـصـة بشكل خاص لكتابة components بصرية التى ليس لها handles متعلقة بها، ولذلك تأخذ حيزاً أقل فى الذاكـرة عن الـ components التى تعـمل كـواجـهـات تطبـيق لعناصـر الـ Windows المعيـارية مثل الأزرار الـ check boxes . وتعـرف الـ Controls unit (وليس الـ Graphics) الـ TGraphicControl .

الرسم بالـ components:

لرسم الجرافيك، يمكنك استدعاء الـ Canvas methods وأضف أحد الـ components التالين على الـ form أو حاوية أخرى:

• **Image:** هـذا الـ component يحتـوى علـى Picture object (الـ TGraphic class)، والذى يحتوى على خاصية الـ Graphic (طبقة الـ TGraphic class). لأن الـ TGraphic class تعد السلف للـ TBitmap class والـ TIcon والـ TMetafile، فإن الـ Image object يمكن أن يحمل ويعرض bitmap أو أيقـونة أو ملف كـبـيـر يمكن أن يشـار إليـه على أنه Image.Picture.Graphic. ويمكن أن يتولى الـ Image أيضاً formats ثانوية مثل الـ JPEG المتوفرة فى الـ JPEG unit والـ TJPEGImage class.

• **Shape:** هذا الـ component يرسم صـورة باسـتـدعـاء الـ Canvas methods. استخدم الـ Shape لرسم وتصوير خطوط، ومستطيلات ودوائر وobjects اخرى.

ان الفرق بين هذين الـ components هو ان الـ Image object يحتوى على صور جرافيكية، و الـ Shape object يرسم ويصور object باستخدام الـCanvas الموروثة. استخدم الـImage components أو الـ Shape لإنشاء صور جرافيكية فى وقت التصميم استخدم الـ Canvas للرسم والتصوير فى وقت التشغيل.

لتخزين objects جرافيكية فى الذاكرة، قم بانشاء object من الـTPicture class. يمكنك ايضاً إنشـاء TBitmap، وTIcon ، وTMetafile objects منفردة، ولكن من الافضل دائماً ان تنشئ TPicture object الذى يحتوى على Bitmap، أو ايقونة، أو ملف كبير. على سبيل المثال جرب OnClick الخاص بالـ Button فى القائمة (١٣-٣)، والذى يقوم بتحميل وعرض ملف bitmap كاملاً فى وقت التشغيل دون استخدام components جرافيكية.

القائمة (١٣-٣): جرب OnClick لهذا الـ Button لتحميل وعرض ملف bitmap كاملاً تحت تحكم البرنامج

```
procedure TForm1.BitBtn1Click(Sender: TObject);
var
```

```
    P: TPicture;
  begin
   P := TPicture.Create;
      try
       P.LoadFromFile('C:\Windows\Clouds.bmp');
          Canvas.Draw(0, 0, P.Graphic);
        finally
       P.Free;
        end;
  end;
```

توضح القائمة (١٣-٣) بعض تقنيات الجرافيك الاساسية. متغير الـ P يعتبر object من الـ TPicture ، والذى تم إنشاؤه باستدعاء TPicture.Create كالعبارة الأولى للـ procedure. لضمان السيطرة على object الناتج، فإن كل العبارات التى تستخدم الـ P توضع داخل نظام الـ try ويستدعى قالب finally الـ P.Free. هذا يضمن أنه، اذا حدثت ايه exception داخل نظام الـ try، يتم تنفيذ العبارة الموجودة فى قالب الـ finally. بغض النظر عن أية اخطاء قد تحدث، فإن الذاكرة المشغولة بالـ TPicture مضمون انها سوف تمحى بشكل سليم قبل إنتهاء الـ procedure.

داخل الـ try، يقوم الـ LoadFromFile method الخاص بالـ TPicture بقراءة ملف bitmap (غير اسم الملف اذا اردت). لعرض الصورة، يستدعى البرنامج الـDraw method الخاص بالـ Canvas، والذى يعرفهDelphi كما يلى:

```
procedure Draw(X, Y: Integer; Graphic: Tgraphic);
```

- **X, Y**: احداثيات Client-area المتعلقة بالـ object والتى توفر الـ Canvas. حدد هذه القيم بصفر لعرض الصورة فى الركن الايسر العلوى من الـ Canvas.

- **Graphic**: قم بتمرير TBitmap، أو TIcon، أو TMetaFile لهذه parameter. أو، كما توضح القائمة، عند استخدام TPicture، قم بتمرير خاصية الـ Graphic التابعة.

أو، يمكنك إضافة Image على الـform واستخدام خاصيةالـ Picture له لتحميل وعرض. الـ bitmap، البرمجة التالية تنفيذ نفس مهام القائمة (١٣-٣):

```
with Image1 do
  Picture.LoadFromFile('C:\Windows\Clouds.Bmp');
```

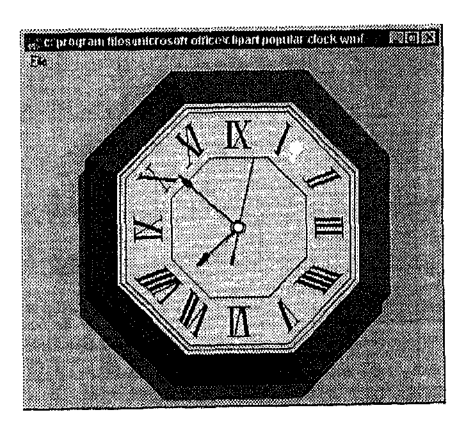
ان الفرق بينهـما هو مجرد اختيار تصميم . وإن كـان من الافضل استخدام component، فإن الـ code السابقة تعتبر الطريقة الأكثر بساطة ، ولا تتطلب منك ان تنشئ وتحرر objects أو تستدعى الـ Draw method لـ Canvas . اذا كنت تحتاج الى إنشاء واستخدام objects فى وقت التشغيل ، استخدم التقنية الموجودة فى القائمة (١٣-٣) . ولكن ، لا تستخدم الـ TPicture الـ window handle ؛ بينما تستخدمه الـ TImage . وهذا يجعل الـ TPicture هى الاختيار الافضل من ناحية الذاكرة .

ملحـوظة: تعتبر الـ class TImage منحدرة من الـ TGraphicControl ، وهى لذلك ليس لديها window handle . ولكن الـ TDBImage لديها window handle ، وهو طرق بين الاثنين .

انظر الباب السابع عشر ، تطوير تطبيقات قاعدة البيانات ، لمزيد من المعلومات حول الـ TDBImage و الـ controls الأخرى لقاعدة البيانات .

Metafile, Bitmap, icon files:

باستخدام المعلومة السابقة ، يمكنك إنشاء متصفح ملف صورة يمكنه ان يعرض ملف حجم كبير (wmf.) ، أو ايقونة (ico.) ، أو bitmap (bmp.) ، وإنسخ أى من هذه الملفات فى دليل آخر . يوضح شكل (١٣-٢) برنامج الـ MetaMore الذى يعرض ملف metafile متوفر مع الـ Microsoft Office . والبرنامج ، الموضح فى القائمة (١٣-٤) ، يمكنه ايضاً ان يقرأ ، ويعرض ، وينسخ bitmap أو Icon .



شكل (١٣-٢): يستطيع الـ MetaMore عرض أى metafile أو bitmap أو ايقونة

القائمة (١٣-٤):MetaMore\Main.pas

```
unit Main;

interface

uses
 SysUtils, Windows, Messages, Classes, Graphics,
    Controls, Forms, Dialogs, Menus, ExtCtrls;
type
    TMainForm = class(TForm)
      Image1: TImage;
         MainMenu1: TMainMenu;
         FileMenu: TMenuItem;
         FileOpen: TMenuItem;
         N1: TMenuItem;
         FileExit: TMenuItem;
         OpenDialog1: TOpenDialog;
         SaveDialog1: TSaveDialog;
         FileSaveAs: TMenuItem;
         procedure FileOpenClick(Sender: TObject);
         procedure FileSaveAsClick(Sender: TObject);
         procedure FileExitClick(Sender: TObject);
       private
      { Private declarations }
       public
      { Public declarations }
       end;

var
 MainForm: TMainForm;

implementation

{$R *.DFM}

procedure TMainForm.FileOpenClick(Sender: TObject);
```

```
begin
 with OpenDialog1 do
    begin
     if Execute then
         Image1.Picture.LoadFromFile(Filename);
           Caption := Lowercase(Filename);
       end;
end;
procedure TMainForm.FileSaveAsClick(Sender: TObject);
begin
 with SaveDialog1 do
    begin
     Filename := Caption;
        if Execute then
             Image1.Picture.SaveToFile(Filename);
     Caption := Lowercase(Filename);
       end;
end;

procedure TMainForm.FileExitClick(Sender: TObject);
begin
 Close;
end;

end.
```

Caution

تحذير: لا يستطيع الـ MetaMore ترجمة انواع الملفات. على سبيل المثال، اذا قمت بتحميل bitmap، يجب ان تحفظها bitmap.

يستخدم الـ MetaMore الـ Image component object ، الذى يعتبر دائماً اسهل الطرق. وبالتبادل، يمكنك إنشاء Picture object كما توضح القائمة (١٣-٣). أو، يمكنك إنشاء object لنوع معين من ملف الصورة، على سبيل المثال، توضح القائمة (١٣-٥) OnClick للـ Button الذى ينشيء TMetaFile ويحمل ويعرض metafile.

اذا كان لديك Microsoft Office، يمكنك ان تجد عينة metafile فى مسار الـ Program Files\Microsoft Office\Clipart\Popular ومسارات دليل أخرى) .

كما هو الحال مع الـ TPicture class، هذه التقنية تحتاج إلى برمجة أكثر من استخدام الـ Image، ولكنها تستخدم ذاكرة اقل Windows Resource أقل . أولاً قم بإنشاء الـ TMetaFile باستدعاء الـ Create method للـ class استخدم للـ object الناتج فى الـ try، وحدده فى قالب الـ finally. هذا يضمن إزالة الـ object من الذاكرة فى حالة حدوث exception .

القائمة (١٣-٥): جرب OnClick لهذا الـ Button لإنشاء TMetaFile وتحميل وعرض metafile

```
procedure TForm1.Button1Click(Sender: TObject);
var
  MetaFile: TMetaFile;
begin
    MetaFile := TMetaFile.Create;
    try
     MetaFile.LoadFromFile(o^C:\msoffice\clipart\anchor.wmf');
       Canvas.Draw(0, 0, MetaFile);
      finally
     MetaFile.Free;
      end;
end;
```

لتحميل metafile، استدع الـ LoadFromFile method الخاص بالـ MetaFile (غير اسم الملف اذا لزم الأمر) . استدع الـ Draw method الخاص بالـ Canvas لعرض الـMetaFile . يمكنك تمرير MetaFile للـ Draw لان الـ TMetaFile class تنحدر من الـ TGraphic .

يمكنك استخدام نفس البرمجة الموجودة فى القائمة (١٣-٥) لتحميل وعرض ملفات ايقونات و bitmap . قم بابدال الـ MetaFile الـ TBitmap أو الـ TIcon، وقم بتحميل النوع المناسب . استخدم الـ TPicture [انظر القائمة (١٣-٣)] لتحميل ملفات الايقونة، أو bitmap، أو الـ metafile حسب امتداد اسم الملف .

بدلاً من الـ Draw ، استـدع الـ StretchDraw لإعـادة تحديد حـجم صـورة لتتناسب داخل مستطيل محدد . على سبيل المثال ، لملء client area النافذة بصورة MetaFile ، استبدل عبارة الـ Canvas.Draw في القائمة (١٣-٥) بهذا السطر :

```
Canvas.StretchDraw(ClientRect, MetaFile);
```

يعرف Delphi الـ StretchDraw على انه :

```
procedure StretchDraw(const Rect: TRect; Graphic: Tgraphic);
```

- **Rect** : يحـدد مـسـاحـة مـسـتطيلة لعـرض صـورة . قـم بتـعـيين قـيـم للاعـداد الصـحـيـحـة Bottom ، Right ، Top ، Left لهـذا السـجل ، أو ، يمكنك تعـيين سجلات TPoint لأعضاء الـ BottomRight والـ TopLeft التابعين للـ Rect .
- **Graphic** : أي TIcon ، أو TBitmap ، أوTMetaFile . الـــــــــــــــــ StretchDraw تعـيـد تحـديد حـجم الصـورة لتـتناسـب داخل المسـاحـة المحـددة للـ Rect.

Bitmap Resoures :

على عكس ما يحدث في برمجة الـ Windows التقليدية ، فإنك لن تستخدم Resources objects عديدة في تطبيـقـات Delphi . وأحد الـ exception لهـذه القاعدة هو الـ bitmap التي تريد ضـمنه لتطبيقاتك . يمكنك توزيع ملف bitmap منفـصل وتسـتـخـدم الـ TBitmap أو الـ TPicture لعـرضـه . أو ، يمكنك إنشـاء bitmap كـ Resource (.res) باسـتـخـدام الـ Image Editor الخاص بـ Delphi على قائمة الـ Tools .

بعـد إنشـاء ملف الـ Resource مع bitmap (ولنفـتـرض انك سـمـيـتـه YOURBITS) ، اربط Resource في ملف exe code. بادخـال هـذا الأمـر في تنفيذ الـ module :

```
{$R Yourbits.Res}
```

استخدم البرمجة الموجودة في القائمة (١٣-٦) لإنشـاء bitmap object ، واستدع الـ LoadFromResourceName الخاص بالـ TBitmap . قم بتـمرير الـ handle الحالي للتطبيق (HInstance) واسم الـ resource الذى سيـتم تحـميله .



وقد يكون bitmap ذات ٢٥٦ لوناً. ويعتبر الـ string YOURBITS هو اسم الـ resource الذى قمت بتعيينه فى الـ Image Editor. (ويمكن ان يكون لملف الـ Yourbits.res اسماً آخر-فلا يجب ان يكون هو نفسه اسم الـ resource الذى يحتويه). استدع الـ Draw methods أو الـ StretchDraw كما هو موضح فى القائمة.

القائمة (١٣-٦): OnClick لهذا الـ Button يوضح كيفية تحميل وعرض الـ bitmap الـresource المرتبط بملف code .exe للبرنامج بواسطة الـ Image Editor الخاص بـ Delphi

```
procedure TForm1.Button1Click(Sender: TObject);
var
  YourBits: TBitmap;
begin
  YourBits := TBitmap.Create;
    try
      YourBits.LoadFromResourceName(HInstance,'YOURBITS');
        Canvas.Draw(0, 0, YourBits);
      (*   Canvas.StretchDraw(ClientRect, YourBits); *)
     finally
         YourBits.Free;
     end;
   end;
```

offscreen bitmap:

توفر الـ TBitmap خاصية Canvas التى يمكنك استخدامها. قم بإنشاء TBitmap ، واستدع الـ methods فى خاصية الـ Canvas التابعة لرسم الـ off Screen له للرسم على الـ Canvas لكى يبقى مختفياً عن الانظار. يمكنك عندئذ عرض الجرافيك الناتج مرة واحدة باستدعاء methods مثل الـ Draw، أو الـ StretchDraw، الـ CopyRect، والـ BrushCopy لـ Canvas تابع لـ form أوالـ PaintBox . هذه التقنية تعتبر هامة فى الصور المتحركة، وكذلك لإخفاء تفاصيل تكوين صورة معقدة حتى يكتمل الشكل.

توضح القائمة (١٣-٧) التقنيات الاساسية المطلوبة للرسم فى bitmap offscreen . لتجربة الـ code استخدم القائمة كأنها Button OnClick. يرسم



البرنامج شكل مستطيل أو بيضاوى بعيداً عن الشاشة ثم يعرض الصورة الناتجة باستدعاء الـ CopyRect .

فى القائمة (١٣-٧)، تقوم سجلات الـ Source والـ Dest TRect بتحديد حجم bitmap offscreen ، وكذلك المكان الذى سيتم فيه نسخ الصورة على الـ form . قم بإنشاء bitmap object باستدعاء الـ TBitmap.Create . عين قيم الـ Width والـ Height لإنشاء bitmap بالحجم الذى تحتاجه .

لرسم الـ offscreen، استدع methods مثل الـ Rectangle والـ Ellipse، وعين قيم لخصائص الـ Pen والـ Brush الاساسية والفرعية . تأكد من استدعاء الـ methods والقيام بالتعينات للـ Canvas الخاص بالـ bitmap . عندما تنتهى من إنشاء صورتك، استدع الـ Canvas method، التابعة للـ PaintBox أو الـ form، مثل الـ CopyRect لإظهار الـ offscreen image .

ان استدعاء الـ TBitmap.Create ينشئ memory device context، والذى يستهلك وقت الـ Windows resources . فى هذا المثال والذى فيه ضغطة زر بسيطة، ستكون النتيجة النهائية سريعة بدرجة كافية، ولكن للحصول على افضل سرعة من خلال loop أو OnPaint ، إنشئ bitmap object فى OnPaint event handler الخاص بالـ form ودمره فى الـ OnDestroy .

عند استخدام عبارات with ، كن حريصاً على ان ترسم داخل الـ Canvas المستهدفة . لرسم الـ offscreen، استدع للـ Canvas methods الخاص بالـ TBitmap . انسخ offscreen ، استدع للـ Canvas methods التابعة للـ PaintBox أو لـ form .

:Drag-and-drrop objects

عندما تصمم تطبيق للـ Delphi ، فإنك تضغط وتسحب component objects على الـ form . ولكن كيف تستطيع ان تفعل نفس الشئ مع objects الجرافيك فى تطبيقاتك والاجابة تتطلب بعض البرمجة والتى لايصعب فهمها، ولكنها لا تتطلب انتباه خاص للتفاصيل خاصة فى التعامل مع احداثيات الفأرة .

القائمة (١٣-٧): OnClick الخاص بهذا الـ Button
يوضح كيفية الرسم داخل الـ offscreen bitmap

```
procedure TForm1.BitBtn1Click(Sender: TObject);
var
 OffScreen: TBitmap;
    Dest, Source: TRect;
begin
 Source := Rect(0, 0, 255, 255);
    Dest := Rect(10, 10, 265, 265);
    OffScreen := TBitmap.Create;
    try
      OffScreen.Width := Source.Right + 1;
         OffScreen.Height := Source.Bottom + 1;
         with OffScreen.Canvas do
         begin
           Pen.Color := clRed;
              Brush.Color := clAppWorkSpace;
              Rectangle(0, 0, 255, 255);
              Ellipse(63, 63, 127, 127);
            end;
         Canvas.CopyRect(Dest, OffScreen.Canvas, Source);
       finally
      OffScreen.Free;
    end;
end;
```



على القرص المدمج: كمثال على الـ drag-and-drop objects الجرافيكية ، جرب تطبيق الـ DragMe على القرص المدمج .

توجد الملفات فى دليل الـ Source\DragMe . كما هو موضح فى الشكل (١٣-٣)، يعرض البرنامج ثلاثة objects : مستطيل دائرى اصفر ، دائرة حمراء ، ومربع ازرق . يعتبر كل منها object هو من الـ Shape component object . قم بتشغيل البرنامج واضغط واسحب الـ objects حول نافذة التطبيق ، توضح القائمة (١٣-٨) الخطوات اللازمة لإنشاء هذه الخدع والذى يمكنك تشغيلها بمعنى أن أى object ترسمه قابل للسحب .

شكل (١٣-٣): يعرض تطبيق الـ DragMe ثلاثة Shapes، التي يمكنك ضغطها وسحبها وتحريكها حول نافذة البرنامج

القائمة (١٣-٨): DragMe\Main.pas

```
unit Main;

interface

uses
  SysUtils, Windows, Messages, Classes, Graphics,
    Controls, Forms, Dialogs, ExtCtrls, StdCtrls, Buttons;

type
    TMainForm = class(TForm)
     Shape1: TShape;
        Shape2: TShape;
        Shape3: TShape;
        BitBtn1: TBitBtn;
        procedure ShapeMouseDown(Sender: TObject;
          Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
            procedure ShapeMouseMove(Sender: TObject;
          Shift: TShiftState; X, Y: Integer);
            procedure ShapeMouseUp(Sender: TObject;
          Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
            procedure FormCreate(Sender: TObject);
      private
     Dragging: Boolean;        // Drag operation in progress
```

```
      XOffset, YOffset: Integer; // Offsets from shape upper
      left
        FocusRect: TRect;        // Dotted outline while
      dragging
        PS: TShape;             // Reference to shape dragging
       public
      { Public declarations }
       end;

var
 MainForm: TMainForm;

implementation

{$R *.DFM}

procedure TMainForm.ShapeMouseDown(Sender: TObject;
    Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
   begin
    Dragging := True;    { Set dragging flag true }
      XOffset := X;       { Keep offsets from shape upper left
     }
      YOffset := Y;
      PS := Sender as TShape;  { Assign reference to shape }
      with PS do             { Create outline rectangle }
     FocusRect := Rect(Left, Top, Left + Width, Top + Height);
      Canvas.DrawFocusRect(FocusRect);  { Draw outline }
end;

procedure TMainForm.ShapeMouseMove(Sender: TObject;
 Shift: TShiftState; X, Y: Integer);
begin
 if Dragging then  { Move outline only if dragging }
    begin
```



```
        Canvas.DrawFocusRect(FocusRect);  { Erase outline }
          with FocusRect do
          begin  { Move outline rectangle }
         Left := (PS.Left + X) _ XOffset;
              Top := (PS.Top + Y) _ YOffset;
              Right := PS.Width + Left;
              Bottom := PS.Height + Top;
            end;
          Canvas.DrawFocusRect(FocusRect);
         end;
end;

procedure TMainForm.ShapeMouseUp(Sender: TObject;
 Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
 if Dragging then  { Move shape only if dragging }
    begin
      Canvas.DrawFocusRect(FocusRect);
        Dragging := False;
        with Sender as TShape do
        begin  { Move shape to new location }
          Left := (Left + X) _ XOffset;
            Top := (Top + Y) _ YOffset;
           end;
         end;
end;

procedure TMainForm.FormCreate(Sender: TObject);
begin
 Dragging := False;
end;

end.
```

يعرف الـ DragMe خمسة متغيرات فى الـ module form class. تكون الـ Dragging محددة بـ True اثناء عملية السحب والاسقاط. يعتبر الـ XOffset والـ YOffset هما احداثيات الـ pixel لـ cursor الفأرة المرتبطة بالـ Shape الذى يتم سحبه. هذه القيم تجعل من السهل "انتقاء" الـ object بضغط الفأرة فى أى مكان داخل حدودها. الـ FocusRect هو سجل TRect ذا قيم Left، وTop، و Right، وBottom التى تحدد المستطيل الخارجى الذى تراه اثناء ضغط وسحب object ما. والـ PS هى pointer للـ objects، والذى يستخدمها البرنامج للإشارة للـ objects الذى يتم سحبه.

تقوم ثلاثة من الـ event handlers بتنفيذ السحب والاسقاط لأى Shape. عندما تضغط cursor الفأرة داخل الـ Shape، يبدأ الـ procedure ShapeMouseDown عملية السحب والاسقاط أولاً، يحدد البرنامج الـ Dragging flag بـ True. ثم يحفظ قيم احداثيات الـ x- والـ y- للفأرة فى الـ XOffset والـ YOffset. هذا يحدد مكان الفأرة بالنسبة للركن الايسر العلوى للـ object. يحفظ البرنامج ايضاً الـ Sender parameter، الذى تم إلقاه الى الـ TShape object، فى الـ PS كـ reference فى المستقبل للـ object الذى تم سحبه. باستخدام الـ PS، يعين البرنامج قيم للـ FocusRect لرسم الـ outline المنقوط خلال السحب بهذه العبارة:

```
Canvas.DrawFocusRect(FocusRect);  { Draw outline }
```

DragFocusRect يرسم outline مستطيل منقوط باستخدام منطق الـ OR. باستدعاء الـ DragFocusRect مرتين لنفس سجل الـ TRect يتم مسح الـ outline. عند ما تحرك الفأرة، اذا كان الـ Dragging محدد بـ True، فإن OnMouseMove، التابع للـ ShapeMouseMove، يستدعى الـ DragFocusRect لمسح المستطيل فى موقعة القديم. يعين البرنامج عندئذ قيم جديدة لاعضاء الـ Left، الـ Top، الـ Right، والـ Bottom التابعيين للـ FocusRect. عند قراءة هذا الـ code، تذكر ان parameters احداثيات الفأرة x و y مرتبطة بالـ objects الذى يتم سحبه، وليس بالنافذة أو الشاشة. وبذلك، فإن موضع الـ FocusRect الجديد يساوى مكان الـ object (والذى لم يتغير) زائد الـ x

و الـ y، ناقص الـ offsets من الركن الايســـر العـلوى للـ object للمكان الذى ضغطت فيه الفأرة. ان استدعاءالـDragFocusRect مع هذه القيم الجديدة ينقل outline عند ما تسحب الفأرة.

عندمـــا تطلق زر الفـــأرة، فـــإن الـ event handler الثـــالث والاخـــيـــر ShapeMouseUp، يحرك الـ object الى مكانه الجديد. (يجب التأكد من أن الـ Dragging مـــحــــدد بـ True، فى هذه الحـــالة ليس ضـــروريـاً أن يســبـــقـــه الـ OnMouseUp event، ولكن من الأفـضل دائمـاً ان تكون حذراً). مـرة اخرى، استدعى البرنامج الـ DragFocusRect، هذه المرة لرسم الـoutline النهائى. بعد تحديد الـ Dragging flag بـ False ، فـمن السهل ان تحرك الـ object بتعيين قيم جـديدة لخـصـائص الـLeft، الـ Top له، مكـمـلاً الخـدع الخـاصـة بالـobject الى الموضع الأخير المعروف للمستطيل الذى تم تحديده.

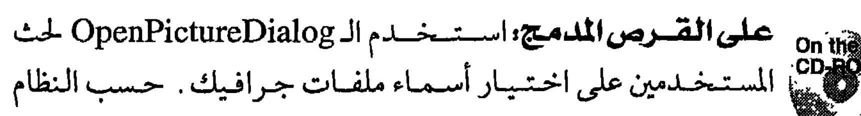
تذكـر ان الـobject يمكن ان تتشـارك فى الـ event handlers. على سـبـيل المثال، لإضافة object جـديد قابل للسحب للـ DragMe، أضـف Shape وحدد الـOnMouseDown، والـOnMouseMove ، والـ OnMouseUp له بالـ event handlers الموجودة للـ module.

:Picture Dialogs

يمكنك اسـتـخـدام dialog components معـيـارية كالـ OpenDialog والـ SaveDialog للتصفح أو لحث المستخدمين على اسماء ملفات جرافيكية. ولكن، هناك components جــديدان نســبــيـــاً، وهمـــا الـ OpenPictureDialog والـ SavePictureDialog، يوفـر ان dialogs خـاصـيـة لتـصـفـح ملفـات الجرافيك. تعرض الـ dialogs صور الجرافيك بحيث يستطيع المستخدمون رؤية الملفـات التى يخـتـارونهـا قـبل تحـمـيلهـا فى تطبيق. و تسـمى فى بعض الأحـيـان thumbnail sketch، بالرغم من اننى لا ارى معنى لهذه التسمة).

الـOpenPictureDialog:



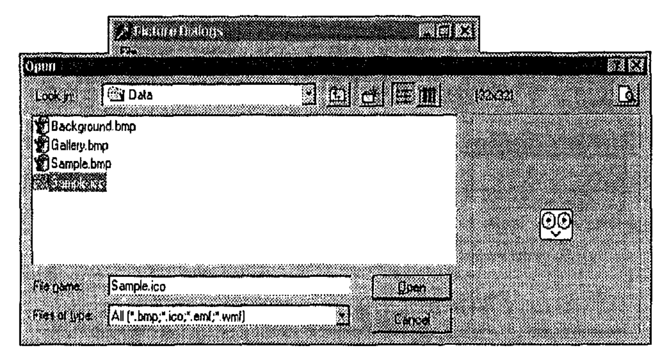


على القـــرص المدمج: اســتـــخـــدم الـ OpenPictureDialog لحث المسـتـخـدمين على اخـتـيـار أسـمـاء ملفـات جـرافـيك. حسب النظام



الافـتـــراضى ، يوضح الـ dialogs كل ملفـــات الـbmp. (bitmap) ، الـ ico. (ايقـــونة) ، الـ enhanced meta file .emf ، والـ meta file .wmf فى الدليل المختار .

عندما يختار المستـخدم واحداً من انواع الملفات هذه ، يعرض الـ dialog الصورة . [انظر شكل (١٣-٤) . يعرض الـ object يوضح الـ dialog الشكل الذى تم عـــرضـــه بواسطة التطبـيق PicDialogs على القـــرص المدمج فى دليل الـ Source\PicDialogs . توضح القائمة (١٣-٩) الـ source code. للبرنامج .



شكل (١٣-٤) : الـ OpenPictureDialog يحث المستخدمين على اختيار اسماء ملفات الجرافيك ، ويعرضها

القائمة (١٣-٩) : PicDialogs\Main.pas

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics,
Controls,
  Forms, Dialogs, Menus, ExtDlgs, ExtCtrls;

type
  TMainForm = class(TForm)
    Image1: TImage;
      OpenPictureDialog1: TOpenPictureDialog;
```

```
    SavePictureDialog1: TSavePictureDialog;
      MainMenu1: TMainMenu;
      File1: TMenuItem;
      Open1: TMenuItem;
      Save1: TMenuItem;
      N1: TMenuItem;
      Exit1: TMenuItem;
      procedure Open1Click(Sender: TObject);
      procedure Exit1Click(Sender: TObject);
      procedure Save1Click(Sender: TObject);
  private
    { Private declarations }
     public
    { Public declarations }
     end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}

procedure TMainForm.Open1Click(Sender: TObject);
begin
  with OpenPictureDialog1 do
  if Execute then
  begin
   Image1.Picture.LoadFromFile(Filename);
     Caption := Lowercase(Filename);
   Save1.Enabled := True; // Enable File|Save... command
    end;
end;

procedure TMainForm.Save1Click(Sender: TObject);
begin
```



Converted by Tiff Combine - (no stamps are applied by registered version)


```
    with SavePictureDialog1 do
     begin
      Filename := Caption;
        if Execute then
        begin
   Image1.Picture.SaveToFile(Filename);
           Caption := Lowercase(Filename);
          end;
       end;
  end;

  procedure TMainForm.Exit1Click(Sender: TObject);
  begin
     Close;
  end;

  end.
```

لاستخدام الـ OpenPictureDialog ، أضف الـ component على forme ثم قم ببرمجة الـ event handler مثل الـ Open1Click فى القائمة والذى يتم تنفيذه عندما يختار المستخدم امر الـ Open الخاص ببرنامجك- من قائمة ، مثلاً أو بضغط زر . فى هذه النقطة ، استدع امر الـ Execute للـ OpenPictureDialog ، والذى يرجع بـ True فقط اذا اختار المستخدم ملف جرافيك . فى اغلب الحالات ، تستخدم code مثل التالية :

```
with OpenPictureDialog1 do
if Execute then
begin
 // Use Filename property here to load file
end;
```

ان عبارة الـ with تخبر الـ Delphi ان يستخدم خصائص وmethods الـ OpenPictureDialog حسب النظام الافتراضي . وعبارة الـ if تستدعى الـExecute ، والذى ، اذا كان True ، ينفذ العبارات الواقعة بين الـ begin والـ end . إدخل برمجك هنا لإستخدام خاصية الـ Filename ، التى تحمل اسم المسار

للملف المختـار . اذا رجعت الـExecute بـ False، فإن المستخدم قد ضغط زر الـ Cancel أو ضغط Esc لإنهاء الـ dialog، ولا يجب ان يقوم برنامجك بتحميل ايه ملف .

خصائص الـ OpenPictureDialog:

فيما يلى بعض الخصائص التى يمكنك تعديلها للتعامل مع الـ OpenPictureDialog object :

- **DefaultExt:** حدد هذه الخاصية بامتداد اسم الملف الافتراضى، دون نقطة، مثل bmp أو .ico . هذا الـ string يتم إلحاقه باسم ملف مختار، الا اذا كان اسم الملف هذا منتهياً بالفعل بامتداد مسجل . اذا اختار المستخدم ملف له امتداد غير مسجل (على سبيل المثال، يكتب المستخدم Anyfile بدلاً من Anyfile.bmp)، فإن الـ DefaultExt يتم إلحاقها باسم الملف . يمكنك ايضاً تحديد الـ DefaultExt فى code بامتداد الجرافيك الافتراضى لـ class الجرافيك مثل الـ TBitmap، باستخدام عبارة مثل :

```
OpenPictureDialog1.DefaultExt := GraphicExtension(TBitmap);
```

- **Files:** اذا كانت خاصية الـ Options تتضمن خيار الـ ofAllowMultiSelect، إذن تحتوى خاصية الـ Files على قائمة string بكل الملفات المختارة . والـ Files.Count تساوى عدد الملفات المختارة .

- **Files:** هذه تعمل تماماً مثل خاصية الـ Files فى dialogs فتح الملف (وحفظ الملف) الأخرى . اضغط الزر البيضاوى التالى لهذه الخاصية لتعديل الصفة الافتراضية للـ file filters، والتى هى حسب النظام الافتراضى، محددة بالقائمة التالية من الـ strings فى الجدول (١٣-١) . والتحديدات الافتراضية مسجلة بالـ TGraphic class . يمكنك الإضافة اليها باستدعاء الـ TPicture.RegisterFileFormat .

- **FilterIndex:** حدد هذه بفهرس الـ string فى filter string-list التى تريد ان يستخدمها البرنامج حسب النظام الافتراضى . هذا هو نوع اسم الملف الذى يراه المستخدمون عندما يفتحون الـ dialog لأول مرة . على سبيل المثال، ان

استخدام الـ filters الافتراضية الموجودة فى الجدول السابق، وتحديد الـ FilterIndex باثنين يؤدى الى اختيار الـ(*.ico) Icons كاسم افتراضى. اذا كان الـ FilterIndex خارج المدى، يتم استخدام الـ Filter string الأول حسب النظام الافتراضى.

● **InitialDir**: حدد هذه باسم المسار الدليل الذى تريد ان يراه المستخدمون عندما يفتحون الـ dialog لأول مرة. اذا لم تحدد هذا الحقل، فإن الدليل الحالى يتم استخدامه حسب النظام الافتراضى.

● **Options**: يمكنك اختيار خيارات متعددة بتعيين قيم لهذه المجموعة. على سبيل المثال، ان استخدام على سبيل المثال، الـ ofAllowMultiSelect لتسمح للمستخدمين باختيار أكثر من اسم ملف واحد (انظر خاصية الـ Files). سوف اوضح المزيد عن الـ Options فى الفصل التالى.

● **Title**: عين الـ string الذى تريد للـ dialog بأن يعرضه فى الـ title bar للنافذة. اذا لم تعين string لهذه الخاصية، يكتب الـ dialog كلمة Open على الـ Title. (يعرض الـ SavePictureDialog كلمة Save).

الجدول (١٣-١): خصائص الـ Filter الافتراضية للـ OpenPictureDialog والـ SavePictureDialog

Filter name	*Filter*
All (*.bmp;*.ico;*.emf;*.wmf)	*.bmp;*.ico;*.emf;*.wmf
Bitmaps (*.bmp)	*.bmp
Icons (*.ico)	*.ico
Enhanced Metafiles (*.emf)	*.emf
Metafiles (*.wmf)	*.wmf

خيارات الـ OpenPictureDialog:

إن الـ OpenPictureDialog والـ SavePictureDialog يقدمان خيارات عديدة يمكنك اختيارها لكى تخصص ظهور الـ dialog وعمله. والمجموعة الكاملة للخيارات يتم تعريفها من نوع Pascal enumerated data:

```
TOpenOption = (ofReadOnly, ofOverwritePrompt,
ofHideReadOnly,
 ofNoChangeDir, ofShowHelp, ofNoValidate,
   ofAllowMultiSelect,
    ofExtensionDifferent, ofPathMustExist,
   ofFileMustExist,
    ofCreatePrompt, ofShareAware, ofNoReadOnlyReturn,
    ofNoTestFileCreate, ofNoNetworkButton,
   ofNoLongNames,
    ofOldStyleDialog, ofNoDereferenceLinks);
TOpenOptions = set of TOpenOption;
```

فى واقع الأمر إنهـا تعـريفـان . الـ TOpenOption (مفـردة) هى النوع الـ enumerated- يمكن تعيين أى من الثوابت المذكورة لمتغير من هذا النوع . والـ TOpenOption (جمع) يتم تعريفه كثوابت الـ TOpenOption لمجموعة Pascal . ويمكن تعيين واحداً أو أكثر أو لا شئ من مجموعات ثوابت الـ TOpenOption لمتغير من نوع الـ TOpenOption . على سبيل المثال ، لتحديد خيارات الـ ofReadOnly والـ ofAllowMultiSelect ، يستطيع البرنامج أن ينفذ عبارة مثل :

```
OpenPictureDialog1.Options := [ofReadOnly,
ofAllowMultiSelect];
```

لتعيين خيارات فى وقت التصميم ، اختر الـ OpenPictureDialog object واضغط مرتين علامة الزائد الصغيرة الى اليسار من ناحية الـ Object فى الـ Object Inspector . هذا يفتح القيم الفرعية لتلك الخصائص ، والتى هى مبرمجة كخصائص True/False مقابلة لثوابت الـ TOpenOption enumerated . على سبيل المثال ، لتقوم بالتعيين السابق فى وقت التصميم ، حدد خصائص الـ ofReadOnly والـ ofAllowMultiSelect بـ True .

الجدول (١٣-٢) يصف كل خيار بالترتيب الأبجدى . لا توجد خيارات خاصة بالـ OpenPictureDialog أو SavePictureDialog الجديد .



جدول (١٣-٢): خيارات الـ OpenPictureDialog والـ SavePictureDialog

الخيــــار	الوصـــــــف
ofAllowMultiSelect	يسمح بإختيار ملفاً واحداً أو أكثر . ويتم إدخال اسماء الملفات المختارة فى خاصية الـ Files . ويساوى الـ Files.Count عدد اسماء الملفات المختارة
ofCreatePrompt	اذا أدخل المستخدم اسم ملف غير موجود، يقدم هذا الخيار dialog يسأل عما اذا كان يجب إنشاء ملف جديد بهذا الاسم. يجب أن يقوم الـ code بإنشاء الملف الفعلى- هذا الخيار يبعث التساؤل فقط، ولكنه لا ينشىء الملف
ofExtensionDifferent	هذا الخيار يعمل كـ flag إما بـ True أو False والذى يشير الى اذا ما كان امتداد اسم الملف مختلف عن الصفة الموجودة فى خاصية الـ DefaultExt. استخدم عبارة مثل : if TOpenOption.ofExtensionDifferent in OpenPictureDialog1.Options then. لتفحص ما اذا كان هذا الـ flag متضمن فى مجموعة الـ Options .
ofFileMustExist	استخدم هذا الخيار لتجعل الـ dialog يعرض رسالة خطأ إذاً حاول المستخدم إدخال اسم ملف غير موجود. هذا الخيار يضمن أن خاصية الـ Filename المدخلة تساوى اسم ملف تم اختياره من قائمة الدليل
ofHideReadOnly	إن dialogs الصورة تكون لديها عادة Open As Read Only check box .
ofNoChangeDir	اذا أردت إعادة تحديد الدليل بمساره قبل أن يتم استدعاء عبارة الـ Execute الخاصة بالـdialog، ليحتوى هذا

	الخيار. اذا لم يحتويه فإن الدليل الحالى يتغير الى أى مسار يختاره المستخدم.
ofNoDereference Links	فى العادى، تكون اختصارات الـ Windows مترجمة الى المسارات الفعلية للملفات التى تشير إليها الاختصارات. اذا لم تكن تريد ان يحدث هذا فتحتوى هذا الخيار. فى هذه الحالة، فإن الـ Filename الذى يتم ادخاله لاختصار اسم مختار سيتم تحديده لملف الـ lnk. الخاص بذلك البند، وان تحديد الملف المشار إليه فعلياً امر يرجع إليك.
ofNoLongNames	يبطل اسماء الملفات الطويلة، ويعرض فقط اسماء النمط القديم ٨,٣ (ثمانية رموز لاسم الملف، ثلاثة رموز الامتداد) فى dialog directory pane.
ofNoNetwork Button	هذا الخيار يحذف زر الـ Network من dialog اختيار الملف، والذى يظهر فقط اذا ما احتواه أيضاً خيار الـ OldStyleDialog.
ofNoReadOnly Return	اذا حاول المستخدم ان يفتح ملف قراءة فقط، فإن هذا الخيار يجعل الـ dialog يعرض رسالة خطأ ويرفض إدخال الملف المختار. إذ أن استخدام هذا الخيار يضمن ان ملفاً يمكن كتابته ايضاً (ولكن احترس من ان حالة الملف قد تتغير بواسطة برنامج آخر فى الوقت الحالى)
ofNoTestFileCreate	استخدام هذا الخيار بحذر شديد. يهدم حماية ملف الشبكة متيحاً الفرصة لحفظ الملف فى دليل شبكة مشترك
ofNoValidate	فى العادى، يتم فحص اسماء الملف لأى رمز غير مسموح به استخدم هذا الخيار فقط اذا كانت تلك الرموز مستخدمة نوعاً ما فى اسماء الملفات الفعلية

ofOldStyleDialog	يتحول الى الشكل القديم للـ dialog (أنظر ايضاً الـofNoNetworkButton)
ofOverwritePrompt	لـ dialog الحفظ فقط، يعرض هذا الخيار رسالة خطأ تحذر المستخدمين من انهم قد اختاروا اسم ملف موجود. اذا استخدمت هذا الخيار، يمكنك ان تتأكد من ان المستخدمين قد أجابوا بـ Yes لهذا النص الـ "Overwrite file?"، ويمكنك بأمان ان تكتب للملف المختار حتى اذا كان موجوداً بالفعل
ofPathMustExist	اذا ادخل المستخدم مسار دليل غير موجود، هذا الخيار يعرض رسالة خطأ ويرفض إدخال المسار فى الـ Filename
ofReadOnly	يحدد أن الملفات المحددة بصفة القراءة فقط التى يمكن إدخالها بواسطة الـ dialog فى خاصية الـ Filename
ofShareAware	استخدمه بحذر شديد. هذا الخيار يهدم انتهاكات التشارك فى ملف (المستخدمون يحاولون ان يفتحوا ملفاً يتم استخدامه، مثلاً)، ويسمح بالاختيار من ملفات قد تكون فى اثناء عملية تعديل من قبل مستخدمين أوبرامج اخرى
ofShowHelp	استخدم هذا الخيار لعرض زر الـ Help. وكذلك قم بتعيين قيمة HelpContext لعرض ملف الـ help الخاص بك

الـSavePictureDialog:

On the CD-ROM

على القرص المدمج: ان الـ SavePictureDialog هو تقريباً الـ OpenPictureDialog. وكلا الـ components لهما خصائص وخيارات متطابقة، ولكن بالطبع، ان الـ SavePictureDialog يحث المستخدمين على اسماء ملفات لحفظ ملفات الجرافيك. إن برنامج العرض

PicDialogs الموجود على القرص المدمج فى دليل الـ Source\PicDialogs يظهر كـيـفـيـة اسـتـخـدام الـ SavePictureDialog [راجع شكل (١٣-٤) والقـائمـة (١٣-٩)].

انك تستخدم الـ SavePictureDialog بنفس الطريقة تقريباً التى تستخدم بها الـ OpenPictureDialog. ولكن، قد تريد ان تأخذ خطوة واحدة إضافية كما هو موضح فى الـ code التالية والتى توضح كيفية الاستجابة لزر أو أمر الـ Save الخاص بالبرنامج:

```
With SavePictureDialog1 do
begin
 Filename := Caption;
    if Execute then
    begin
  Image1.Picture.SaveToFile(Filename);
        Caption := Lowercase(Filename);
      end;
end;
```

هذه الـ code، المأخوذة من التطبيق PicDialogs، تستخدم عبارة with لتحدد ان تلك الخصائص والـ methods التى تنتمى للـ SavePictureDialog1 يجب ان يتم استخدامها. ان خاصية الـ Filename لذلك الـ object يتم تحديدها اولاً فى الـ window caption- هذا يجعل الـ dialog يعرض اسم الملف الحالى، الذى تم حفظه فى caption النافذة بواسطة close-command procedure (انظر الـ Open1Click فى القـائمـة). ان اسـتـدعـاء الـ Execute يظهـر الـ dialog، ويدخل True فقط اذا اختار المستخدم اسم ملف وضغط زر الـ Save. فى هذه الحالة، فان الـ SaveToFile method الخاص بالـ Image فى خاصية الـ Picture يتم استدعاءه لكتابة الملف على القرص. ولان المستخدم قد يكون اختار اسم ملف مختلف ليحفظ فيه الملف، فان caption النافذة يتم إعادة تحديده بخاصية الـ Filename الحالية للـ SavePictureDialog1. هذا يضمن ان اسم الملف المختار و caption النافذة متفقان دائماً.

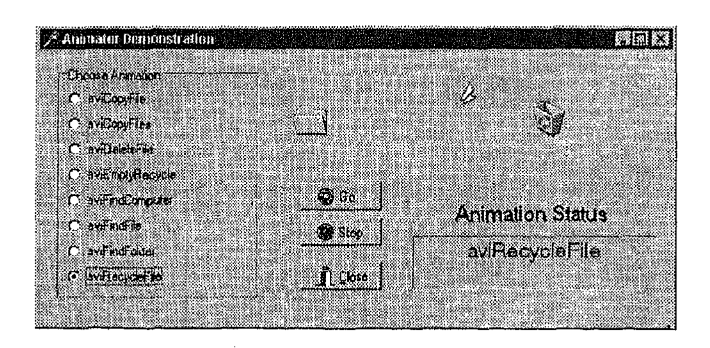
:Animations

مع الـ Delphi ، فإنه من السهل إضافة صور متحركة لبرنامجك . فى هذا الفصل ، سوف اشرح اسس استخدام الـ Animate والـ MediaPlayer . مع الـ Animate ، يمكنك إضافة صور متحركة معيارية تعطى تغذية خلفية بصرية للمستخدمين اثناء عمليات عامة مثل نسخ وحذف الملفات . يمكنك ايضاً ان تلعب بأى ملفavi. (سمعى بصرى) . مع الـ MediaPlayer ، يمكنك إنشاء متصفح ملف avi. ذى خصائص كاملة ، كما يوضح تطبيق الـ VideoPlayer الخاص بهذا الفصل .

الـAnimate component :

على القـــــرص المدمج: ان الـAnimate على الـ Win32 palette يمكن ان يدير أى ملف AVI ليوفر صور جرافيكية متحركة فى النافذة . يمكنك ايضاً استخدام هذا الـ component لعرض قصاصات الـ AVI العامة المتوفرة مع الـ Windows كذلك المعروضة بواسطة الـ Explorer اثناء عمليات الملف .

يوضح شكل (١٣-٥) واحدة من ثمانى صور متحركة معيارية ممكنة معروضة بواسطة التطبيق Animator الموجود على القرص المدمج فى دليل الـ Source\Animator . قم بتحميل ملف مشروع هذا البرنامج ، Animator.dpr ، فى Delphi ، واضغط F9 تشغيل برنامج العرض . اختر واحداً من ازرار الـ radio لترى صورة متحركة مختلفة . اضغط زر الـ Go لتشغيل الصور المتحركة ؛ اضغط Stop للتوقف . توضح القائمة (١٣-١٠) الـ source code لبرنامج الـ Animator .



شكل (١٣-٥): التطبيق Animator يوضح كيفية استخدام الـ Animator لتشغيل AVI معيارية لـ Windows ، فى هذه الحالة، هى صورة الـ "Recycle File"



القائمة (١٢-١٠): Animator\Main.pas

```
unit Main;

interface

uses
 Windows, Messages, SysUtils, Classes, Graphics,
   Controls,
     Forms, Dialogs, StdCtrls, Buttons, ExtCtrls, ComCtrls;

type
  TMainForm = class(TForm)
     Animate1: TAnimate;
        RadioGroup1: TRadioGroup;
        GoBitBtn: TBitBtn;
        StopBitBtn: TBitBtn;
        BitBtn1: TBitBtn;
        StatusText: TStaticText;
        Label1: TLabel;
        procedure GoBitBtnClick(Sender: TObject);
        procedure StopBitBtnClick(Sender: TObject);
       private
  { Private declarations }
       public
  { Public declarations }
       end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}

{ Define array types for easier assignments of the
 Animate object's CommonAvi property, and also the
    StatusText object that shows which animation is
  running. }
type
```

```
    aviKindArray = array[0 .. 7] of TCommonAvi;
     aviStringArray = array[0 .. 7] of String;

{ Define constant arrays containing the TCommonAvi
values
  in the same order they appear in the RadioGroup object,
     and also strings for displaying in the StatusText object
     that shows which animation is running. }
const

  aviKinds: aviKindArray =
   (aviCopyFile,
       aviCopyFiles,
       aviDeleteFile,
       aviEmptyRecycle,
       aviFindComputer,
       aviFindFile,
       aviFindFolder,
       aviRecycleFile);

  aviStrings: aviStringArray =
   ('aviCopyFile',
       'aviCopyFiles',
       'aviDeleteFile',
       'aviEmptyRecycle',
       'aviFindComputer',
       'aviFindFile',
       'aviFindFolder',
       'aviRecycleFile');

{ Start selected animation. This event handler is assigned
  to the OnClick event for both the Go button and the
   RadioGroup1. Clicking Go or clicking a radio button
 starts
   the animation immediately. }
procedure TMainForm.GoBitBtnClick(Sender: TObject);
var
  AnimIndex: Integer;  // Index of selected animation
begin
```



```
AnimIndex := RadioGroup1.ItemIndex;
 with Animate1 do
 begin
  StatusText.Caption := aviStrings[AnimIndex];
 CommonAVI := aviKinds[AnimIndex];
      Play(1, FrameCount, 0);  // Start the animation
    end;
end;

{ Halt the animation when user clicks the Stop button. }
procedure TMainForm.StopBitBtnClick(Sender: TObject);
begin
 Animate1.Stop;
   StatusText.Caption := '(stopped)';
end;

end.
```

يوضح تطبيق الـ Animator كيفية اختيار، بدء، وإيقاف الـ Windows AVClip المعيارية باستخدام الـ Animator . ان الـ GoBitBtnClick procedure يتم تعيينه للـ OnClick لكلاً من زر الـ Go والـ RadioGroup1 وكنتيجة لذلك، فان ضغط الـ Go أو اختيار صورة متحركة من نوع زر الـ radio يؤدى الى بدء تشغيل الصورة المتحركة على الفور.

يبدأ الـ GoBitBtnClick procedure بتعيين الـ ItemIndex الخاص بالـ RadioGroup1 للـ Integer المحلى، حتى يجعل استخدامه اسهل. هذا يشير الى زر المجموعة الذى يتم اختياره حالياً. لتظهر على الشاشة أى صور متحركة يتم تشغيلها، يتم تعيين string للـ StaticText object من constant array، aviStrings معرفة فى قطاع الـ implementation الخاص بالـ unit. باستخدام نفس فهرس مجموعة الـ radio ، فإن خاصية الـ CommonAvi للـ Animate يتم تحديدها باسم الصورة المتحركة المختارة مثل aviCopyFile أو aviFindComputer. لإدارة الصورة المتحركة، يستدعى البرنامج الـ Play method الخاص بالـ Animate باستخدام العبارة:

```
Play(1, FrameCount, 0);
```



ان arguments الأولتين تشيران الى مدى اعداد الاطار الذى يتم إدارتها . فالاطار الاول يحمل الرقم واحد . والـ FrameCount ، وهو حقل فى الـ Animate ، يشير الى عدد الإطارات فى الـ argument AVI Clips. الأخيرة تساوى عدد التكرارات . حدد هذه القيمة بصفر ، كما هو الحال هنا ، لتعيد عرض الصورة المتحركة حتى يتم استدعاء الـ Stop method الخاص بالـ Animate .

هذا يحدث فى الـ Animate عندما تضغط زر الـStop . هذا يؤدى الى استدعاء StopBitBtnClick ، الذى يستدعى الـ Stop method الخاص بالـ Animate . للإشارة الى انه لا يوجد تشغيل لصورة متحركة ، يحدد البرنامج ايضاً الـ Caption الخاص بالـ StatusText بقراءة .'(stopped)' .

Animate properties:

فيما يلى خصائص الـ Animate والتى يمكنك استخدامها لتخصيص الـ Animate :

● **Active**: حدد هذه بـ True لتبدأ تحريك الصورة المتحركة . ولكن قبل تحديد الـ Active بـ True يجب ان تختار AVI Clip معيارية بخاصية الـ CommonAVI ، أو تدخل اسم ملف لصورة الـ AVI فى حقل الـFileName . ان تحديد الـ CommonAVI بـ aviNone ، أو مسح حقل الـ FileName، يحدد الـ Active بـ File . يمكنك ايضاً فحص الـ Active فى وقت تشغيل لتحدد ما اذا كانت الصورة المتحركة فى حالة تشغيل .

● **AutoSize**: حدد هذه عادة بـ True حتى يتم تحديد حجم الـ Animate بصورة تلقائية إعتماداً على حجم الـ AVI Clip. حدد الـ AutoSize بـ File فقط اذا كانت كل القصاصات لها نفس الحجم ، أو اذا كنت تدير Clip واحدة فقط . فكرة : لتحدد كبر حجم الـ Animate الذى تقوم به ، حدد الـ AutoSize بـ True واختر CommonAVI أو FileName . ثم حدد الـ Active و الـ AutoSize بـ File . ان الـ Animate محدد الآن بالحجم المطلوب الـ AVI Clip المختارة .

● **Center**: حدد هذه عادة بـ True بحيث يصبح الصورة المتحركة فى منتصف مسافة عرض الـ Animate . تحديد الـ Center بـ File تجعل الصورة المتحركة تظهر فى الركن الايسر العلوى من الـ Animate .



• **CommonAVI**: هذه الخاصية تختار الـ Windows AVI Clip المعيارية. حـدد الـ CommonAVI بواحـد من الثـوابت المعـدودة لـ TCommonAvi وهي aviNone، aviFindFolder، aviFindFile، aviFindComputer، aviCopyFiles، aviCopyFile، aviRecycleFile، aviEmptyRecycle ، و aviDeleteFile.

• **FileName**: ادخل اسم ملف من نوع ملف الـ AVI Clip - فـقط (لا صوت) في هذه الخاصيـة. يمكنك ان تفعل هذا في وقت التصميم أو في وقت التشغيل. هذا يحدد بصورة تلقائية الـ CommonAVI بـ aviNone - لا يمكنك اختيـار الـ Windows AVI Clip العـامـة وملفـاً في نفس الوقت؛ يمكنك ان تستخدم إحدى الخاصيتين أو الأخـرى. يجب ان يوجد الملف، ويجب ان يكون الـ AVI Clip صـامتة. (انظر الـ "MediaPlayer component" في هذا البـاب لمزيد من المعلومات عن كيفية ادارة افلام و الـ Sound Clip).

• **FrameCount**: يساوي عدد الإطارات في الـ AVI Clip. هذه القيمة هي قـراءة فـقط، فـهي لا تظهـر في الـ Object Inspector. يتم تحـديث الـ FrameCount بصـورة تلقـائيـة عند تحـديد قـيم للـ CommonAVI أو الـ FileName.

• **Repetitions**: تحدد بعدد التكرارات المخصصة لإدارة الـ Clip عندما تكون الـ Active بـ True . وقيمـة الصفـر تشير الى ان Clips يجب ان تدار بلا توقف حـتى يتم تحـديد الـ Active بـ File أو يتم اسـتـدعـاء الـStop method للـ Animate.

• **StartFrame**: تحدد الإطار البادئ لتبدأ الصورة المتحركة. الإطار الأول يحمل الرقم واحد. إنك لا تحتاج ان تحدد هذا الحقل اذا كان برنامجك يستدعي الـ Play method.

• **StopFrame**: عدد الإطار الأخير الذي يتم إدارته في الصورة المتحركة. يوجد عدد الإطارات في حـقل الـ FrameCount قراءة-فقط. إنك لا تحتاج ان تحدد هذا الحقل اذا كان برنامجك يستدعي الـ Play method.



• **Timers**: عندما يحدد بـ false (وهو التحديد الافتراضى)، تؤدى هذه الخاصية بالصورة المتحركة الى التشغيل فى thread منفصل. عند تحديده بـ True، تؤدى هذه الخاصية بالصورة المتحركة الى ان تكون محكومة بالنظام العادى، يمكن ان تترك الـ Timers محددة بـ false، ولكن اذا اردت ان تتزامن events اخرى مع الصورة المتحركة، يمكنك تغيير الـ Timers بـ True. استخدم الـ events OnStart والـ OnStop الخاصة بالـ Animate للتحكم فى نشاط التزامن.

• **Transparent**: حدده بـ True لعرض لون الـobject الأم كخلفية للصورة المتحركة. هذه هى القيمة العادية عند عرض الصورة المتحركة. اذا كانت الـTransparent محددة بـ false، يؤخذ لون خلفية الصورة المتحركة من الـ AVI Clip.

الـMediaPlayer component:

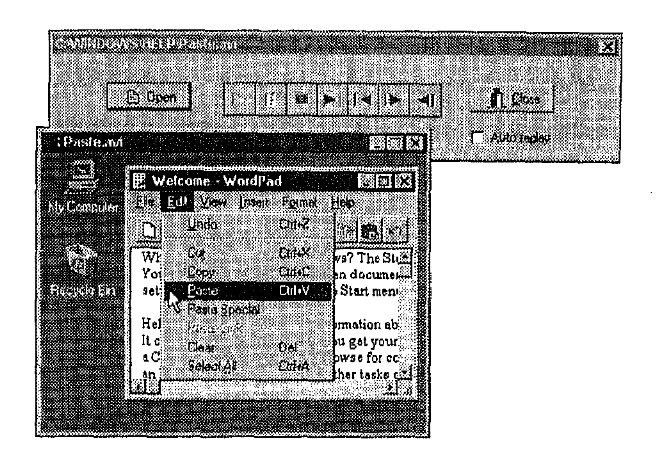
On the CD-ROM **على القرص المدمج**:ان الـ MediaPlayer يعتبر استديو منزلى فعلى يمكنك استخدامه لإنشاء برمجيات الوسائط المتعددة. لقد استخدمت الـ MediaPlayer لإنشاء برنامج العرض VideoPlayer على القرص المدمج (فى دليل الـ Source\VideoPlayer). وهذا البرنامج يستطيع ان يفتح ويدير أى AVI Sound Clip، ويوضح كيفية إنشاء ميزات مثل خيار التكرار الآلى.

يوضح شكل (١٣-٦) برنامج الـ VideoPlayer وهو يدير AVI Clip وجدتها فى دليل الـ Windows\Help- هناك العديد من الـ Clips مخزنة فى هذا المكان وتعتبر جزءاً من نظام online الخاصة بالـ Windows. والـ Clip الموضحة هنا هى صورة توضح القص واللصق. توضح القائمة (١٣-١١) الـ source code للـVideoPlayer. ان بعض خطوات البرنامج لها تأثير خطير على نتائج وقت التشغيل، لذا اضفت العديد من التعليقات فى القائمة لاوضح العبارات. سوف اشرح المزيد عن عمل البرنامج بعد القائمة.

Note **ملحوظة**: ان الـ VideoPlayer هو أكثر من مجرد محرك صورة. فهو يمكن ان يعمل كـcontrol للوسائط المتعددة للقرص المدمج، أو الـ



MIDI ، أو الـ VCR . فى هذا البـاب ، سـوف اشـرح كـيـفـيـة اسـتـخـدام الـ MediaPlayer لعرض AVI Clip صامتة وصوتية ، ولكن نفس الـ component قـادراً على العـمل كـ control للعـديد من انواع الوسـائط المتعـددة . يسـتطيع الـ MediaPlayer ل ايضاً ان يدير افلام الـ MPEG اذا قام المستخدم بتركيب برنامج الـ ActiveMovie الخاص بالـ Microsoft .



شكل (١٣-٦):تطبـيق الـ VideoPlayer على القـرص المدمج يمكن ان يعـرض أى AVI Sound Picture Clip. والـ Clip الموضحة هنا هى Paste.avi من دليل الـ Windows\Help

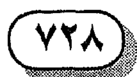
القائمة (١٣-١١): VideoPlayer.Main.pas

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics,
    Controls,
      Forms, Dialogs, StdCtrls, Buttons, MPlayer, ComCtrls;

type
  TForm1 = class(TForm)
      MediaPlayer1: TMediaPlayer;
         OpenBitBtn: TBitBtn;
```

```
        BitBtn2: TBitBtn;
          OpenDialog1: TOpenDialog;
          AutoPlayCheckBox: TCheckBox;
          procedure OpenBitBtnClick(Sender: TObject);
          procedure MediaPlayer1Notify(Sender: TObject);
          procedure MediaPlayer1Click(Sender: TObject;
            Button: TMPBtnType; var DoDefault: Boolean);
          procedure AutoPlayCheckBoxClick(Sender:
        TObject);
      private
        { Private declarations }
          public
        { Public declarations }
          end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

(* Note: Set the MediaPlayer1 object properties as follows
      AutoEnable       False    All buttons always enabled
         AutoOpen         True     Doesn't really matter
         AutoRewind       True     Rewinds media when it
     stops
         Other properties  Default settings
*)
{ Responds to user selection of the Open file button. }
{ Opens media file and starts playing immediately. }
{ Also sets the window caption to the filename. }
procedure TForm1.OpenBitBtnClick(Sender: TObject);
begin
    if OpenDialog1.Execute then
    begin
```



```
      Form1.Caption := OpenDialog1.FileName;
        MediaPlayer1.FileName :=
      OpenDialog1.FileName;
        MediaPlayer1.Notify := True;  // Wants media
      event notify
        MediaPlayer1.Open;          // Opens assigned file
        MediaPlayer1.Frames := 1;   // Sets single step
      frames
        MediaPlayer1.Play;          // Start playing the file
    end;
end;

{ Responds to media event notifications. This gives us the
    chance to check whether the auto-replay checkbox is
  set, and
    if so, and if the media is stopped, to restart play.
  Despite
    the control's documentation, it is necessary to check
  whether
    Notify is true. Even if this flag is false, the procedure is
    called when the user clicks the stop button. }
procedure TForm1.MediaPlayer1Notify(Sender: TObject);
begin
    if (MediaPlayer1.Notify) and          // If flag is true
      (MediaPlayer1.Mode = mpStopped) and  // and playis
      stopped
        (AutoPlayCheckBox.Checked) then    // &
      checkbox enabled
       begin
      MediaPlayer1.Rewind;                 // rewind to start
        MediaPlayer1.Play;                 // and begin
  playing
    end;
   { You must set Notify to True so that the next media
  event
   generates a notification; otherwise, this procedure
  would be called only once. }
      MediaPlayer1.Notify := True;  // Request next
  notification
```

```
end;

{ Responds to user clicking in the MediaPlayer object. We
  need
    to check for this because, if the auto-replay checkbox is
    enabled, stopping the media would generate a
  notification,
    which would start playing again! In other words, this
    procedure allows the media to be stopped regardless of
  the
    auto-replay checkbox setting. }
procedure TForm1.MediaPlayer1Click(Sender: TObject;
  Button: TMPBtnType; var DoDefault: Boolean);
begin
    if (Button = btStop) or (Button = btPause) then
      MediaPlayer1.Notify := False   // Do not continue
  replay
    else
      MediaPlayer1.Notify := True;   // Replay if checkbox
  is set
end;

{ Responds to user changing the state of the auto-replay
  checkbox. If the checkbox is being enabled, this procedure
    turns on notifications so that, when the media stops, the
    Notify event handler can restart the media playing. }
procedure TForm1.AutoPlayCheckBoxClick(Sender:
TObject);
begin
    if AutoPlayCheckBox.Checked then
      MediaPlayer1.Notify := True;  // Triggers notifications
end;

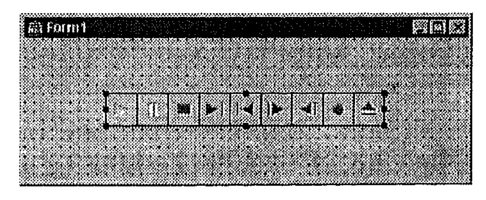
end.
```

هناك العديد من العوامل تتضامن فى إنجاح كتابة محرك صورة AVI مثل التطبيق VideoPlayer المذكور هنا. ان نسخة نظام التشغيل، والآت العرض،



ومحركات الصورة على نظام المستخدم، وformat ملف الـ AVI كل هذه العوامل تؤثر على الناتج. ان نظم الحاسوب الحديثة تعطى افضل النتائج- اما تلك التى لها معدات قديمة ونسخ أولية من الـ Windows 95 قد تعمل بشكل بطيئ أولا تحـرك الـ AVI Clip أصلاً. على ايه حال، ان الـ MediaPlayer يبدو قوياً بدرجة عالية- فقد استخدمت الـ VideoPlayer لتشغيل بعض الـ AVI Clips التالفة والتى لم أكن أراها بأية وسيلة أخرى.

على الشاشة، يعتبر الـ MediaPlayer مجرد toolbar من ازرار يمكن للمستخدمين ان يتعاملوا معها لتشغيل الـ object، مثل الازرار الموجودة على الـ VCR. يوضح شكل (١٣-٧) MediaPlayer غير معدل تم إضافته على form. والازرار التسعة بالترتيب من اليسار الى اليمين هى: Play، Pause، Stop، Next، Previous، Step، Back، Record، وEject. وهناك ثلاث خصائص تؤثر على مظهر الزر. استخدم خاصية الـ ColoredButtons لعرض ازرار منفردة بالالوان أو الابيض والاسود. استخدام الـ EnabledButtons لتشغيل أو إبطال ازرار منفردة (تعرض الازرار المبطلة باللون الرمادى المعتم، ولا يمكن اختيارها). استخدم خاصية الـ VisibleButtons للإشارة الى الازرار التى يجب عرضها. هذه مفيدة فى إخفاء الازرار الغير مرغوب فيها- يقوم الـ VideoPlayer، مثلاً باخفاء ازرار الـ Record والـ Eject التى ليس لها استخدام فى البرامج.



شكل (١٣-٧): MediaPlayer غير معدل بكل ازراره

لبدء إدارة الـ AVI Clip، استخدم الـ dialog box أو وسيلة اخرى لحث المستخدمين على إعطاء اسم ملف. يعرض الـ VideoPlayer كيفية برمجة هذا الفعل فى OpenBitBtnClick، والذى يتم تنفيذه عندما تضغط زر الـ Open:

```
if OpenDialog1.Execute then
```

```
begin
  Form1.Caption := OpenDialog1.FileName;
    MediaPlayer1.FileName := OpenDialog1.FileName;
    MediaPlayer1.Notify := True;  // Wants media event
  notify
    MediaPlayer1.Open;          // Opens assigned file
    MediaPlayer1.Frames := 1;    // Sets single step frames
    MediaPlayer1.Play;          // Start playing the file
end;
```

ان استدعاء الـ Execute يـــؤدى الى إظهـــار نافذة الـ OpenDialog . الـ function ترجع بـ True اذا كان المستخدم قد اختار اسم ملف، فى هذه الحالة يتم تنفيذ العبارات الواقعة بين الـ begin و end . لعرض الملف الذى يتم تشغيله، تحـدد العبـارة الأولى Caption الـ from باسم الملف المختـار، المأخـوذ من الـ OpenDialog1 . نفس هذا الاسم يتم تعيينه لخاصيـة الـ FileName التابعة للـ MediaPlayer1- ربما يكون هذا هو الملف الذى يتم تصفحه . للإشارة الى اننا نريد اخطارات عن events الوسائط، يحدد البرنامج الـ Notify بـ True (المزيد عن هذه النقطة سيأتى فيما بعد).

ثم، يقـوم الـ Open method بتحـميل الملف فى الذاكرة، ولكن لا يقـوم الآن بإدارته . للإشـارة الى كم إطاراً يجب ان يقـوم الزر بتـقـديمهـا للصـورة، يتم تحـديد الـ Frames بواحـد (يتم تحـديد هذه عادة بـ ١٠٪ من عدد الاطارات فى الصورة، ولكننى افضل زيادة إطاراً واحداً فى كل مرة). جدير بالذكر ان خاصية الـ Frames غير منشورة- فيمكنك ان تحددها فى وقت التشغيل، ولكن ليس فى الـ Object Inspector. .

لبـدء إدارة الـ AVI Clip، يقـوم باسـتـدعـاء الـ Play method للـ MediaPlayer1 . على عكس الـ Animate، هذه النسـخة من الـ Play ليس لهـا arguments . يتم استدعائها لتبدأ تشغيل الصورة . اذا لم يقم البرنامج باستدعاء الـ Play، يجب ان يضغط المستخدمون زر الـ Play لبدء تشغيل الـ AVI Clip .

بتحـديد خـاصية الـ Notify بـ True، والتى يتم توليـدها لـ events متنوعة تضع نقطة بدء وإيقـاف الـ Clip، حـدد الـ Notify بـ True (يجب ان تفعل هذا



باستخدام عبارة برنامج كتلك الموضحة فى الجزئية السابقة)، وقم بإنشاء OnNotify للـMediaPlayer . وOnNotify الخاص بالـVideoPlayer هو :

```
procedure TForm1.MediaPlayer1Notify(Sender: TObject);
begin
 if (MediaPlayer1.Notify) and          // If flag is true
     (MediaPlayer1.Mode = mpStopped) and  // and play's
 stopped
       (AutoPlayCheckBox.Checked) then      // &
 checkbox enabled
   begin
    MediaPlayer1.Rewind;              // rewind to start
      MediaPlayer1.Play;              // and begin
 playing
   end;
   MediaPlayer1.Notify := True;  // Request next
 notification
end;
```

ان هدف هذا الـ code هو إعادة إدارة الـ AVI Clip الحالية اذا كان المستخدم قد اختار check box اختيار إعادة الإدارة Auto . ولان الاخطارات المتعددة يتلقاها نفس هذا الـ procedure ، فإن الخطوة الأولى هى لتحديد ما اذا كانت الصورة قد توقفت ، فى هذه الحالة يتم إرجاعها وإعادة بدئها من البداية . اذا لم يتم اختيار AutoPlayCheckBox ، فلا يفعل الـ procedure شيئاً ، ولذلك ، تتوقف الصورة عند نهايتها ، (أو عندما يضغط المستخدم زر الـ Stop) .

بالرغم من ان البرنامج يحدد الـ Notify بـ True ، فمازال ضرورياً ان تختار الـ Notify عند الإدخال للـOnNotify . تشير الاختبارات الى ان الاخطارات مازال يتم تلقيها فى بعض الاحيان عندما تكون الـ Notify بـ False ، ربما لان تلك الـ event يتم توليدها فى الاوقات التى تقوم فيها الـ Notify تغير القيمة . اذا كانت الـ Notify محددة بـ True ، فإن OnNotify الخاص بالـ VideoPlayer يحدد خاصية الـ Mode ليحدد حالة الـ object . اذا كانت هذه الحالة تساوى mpStopped ، إذن يكون المستخدم قد أوقف الإدارة (أو انها قد إنتهت) . اذا كان الـ AutoPlayCheckBox مختاراً ، يقوم البرنامج بارجاع وإعادة إدارة الـ Clip



الحـــاليــة باســـتـــدعـــاء الـ Play method والـ Rewind الخـــاصـــة بالـ MediaPlayer's.

ملحـوظة: لاحظ ان الـ Notify تكون محددة بـ True قبل انتهاء الـ OnNotify مباشرة. هذا ضرورياً لطلب مزيد من الاخطارات. اذا لم تكن تريد المزيد من الاخطارات، حدد الـ Notify بـ False قبل إدخال الـ OnNotify event.

ونصيحة اخيرة بالنسبة للإخطار هي انه عندما يقوم المستخدم بايقاف الصورة، يجب ان يتوقف الفعل فوراً بدلاً من إعادة البدء من جديد. كما هو مكتوب، يكرر الـOnNotify الصورة الحالية اذا كان الـAuto replay مختاراً- هذا قد يؤدى بقاصة الـ AVI Clip للتكرار حتى اذا تم ايقافها بضغط زر الـ Stop الخاص بالـ controller.

لحل هذه المشكلة، يقوم الـ OnClick خاص بالـ MediaPlayer بفحص ما اذا كان المستخدم قد ضغط زر الـ Stop. ها هو الـ procedure الكامل من تطبيق الـ VideoPlaye :

```
procedure TForm1.MediaPlayer1Click(Sender: TObject;
  Button: TMPBtnType; var DoDefault: Boolean);
begin
  if (Button = btStop) or (Button = btPause) then
      MediaPlayer1.Notify := False   // Do not continue
      replay
        else
      MediaPlayer1.Notify := True;   // Replay if checkbox
      is set
end;
```

يتلقى الـ procedure ثلاثة parameters الـ Sender يعتبر الـ object الذى تم ضغطه (فى هذه الحالة هو الـ MediaPlayer1). ولان هناك object واحداً فقط من هذا النوع، فيمكننا ان نتجاهل هذا الـ parameter ونحن مطمئين. يشير الـ Button parameter الى الزر الذى تم ضغطه. وهذا الـ parameter يساوى احدى هذه القيم: btBack ، btEjec، btNext ، btPause ، btPlay ،



btPrev، btRecord ، btStep ، أو btStop . فى الـ code ، يفحص البرنامج ما اذا كان المستخدم قد ضغط ازرار الـ Stop أو الـ Pause . اذا كان كذلك، يتم تحـديد الـ Notify بـ False بـحـيث لا يسـتـمـر تلقى الاخطارات بواسطة OnNotify الخـاص بالـ MediaPlayer ، وبذلك، يتم ابطال الـ auto-replay بغض النظر عن قيمة الـ check box .

والـ parameter الأخيـر، DoDefault، الذى يتلقـاه OnClick الخـاص بالـMediaPlayer يشـيـر اذا مـا كـان الـ code أو الـ component يجب ان ينفـذ اعـمـالاً مـعـيـارية. اذا كان الـ DoDefault مـحـدد بـ True (وهو الافتراضى) ينفذ code لكل الازرار. ان ضغط الـ Play، مثلاً، يجعل الـ MediaPlayer يستدعى الـ Play method الخاص به.

اذا لم تكن تـريد للـ component ان يسـتـدعى الـ methods. الخـاصـة به، حدد الـ DoDefault بـ False فى OnClick التابع للـ MediaPlayer . هذا ينبه الـ component الى ان الـ code قـد نفـذت عـبـارات بديلة لزر أو آخـر. على سـبـيل المثـال، لتنتقل من بين اطارات الـAVI Clip، قد تسـتـخـدم الـ code التاليـة فى الـ OnClick:

```
begin
    if Button = btStep then
    begin
     MediaPlayer1.Frames := 1;
        MediaPlayer1.Step;
        DoDefault := False;
    end;
    // Other buttons execute standard methods
end;
```

اذا كان الـ Button parameter يشير الى ان زر الـ Step قد تم ضغطه، يحدد البـرنامج الـ Frames بواحـد ويسـتـدعى منهج الـ Step للـ MediaPlayer. هذا يجعل الـ Clip تتقدم باطار واحد بدلاً من ١٠٪ من العـدد الكلى للإطارات، وهو التحديد المعيارى. لتنبية الـMediaPlayer الى ان البرنامج قد استجاب بالفعل لزر الـ Step، فإن الـ DoDefault يتم تحديده بـ False. ولان الـ DoDefault محدد بـ



Trueحسب النظام الافتراضى ، فبعد ان ينتهى الـ OnClick ، يستجيب الـMediaPlayer لضغطات زر آخر بشكل عادى .

خصائص الـMediaPlayer:

فيما يلى بعض الملاحظات حول الخصائص المختارة للـMediaPlayer التى يمكنك استخدامهالكى تجعل الـcomponent object متفق مع رغباتك :

- **AutoEnable:** حدد بـ True لتشغيل وابطال ازرار الـ MediaPlayer المختارة إعتماداً على حالة جهاز الوسائط المتعددة . على سبيل المثال ، عندما تحدد هذه الخاصية بـ True ، يتم ابطال زر الـ Play عندما لا يتم تحميل ملف . حدد الـ AutoEnable بـ False لتشغيل كل الازرار طوال الوقت . ان الـ VideoPlayer يستخدم هذا التحديد .

- **AutoOpen:**حددها بـ True لفتح ملف بصورة تلقائية تم تحديده من جانب الـ FileName . اذا كانت محددة بـFalse، يجب ان تستدعى الـ Open method لفتح الملف .

- **AutoRewind:** حددها بـ True لإرجاع ملف الوسائط المتعددة عندما ينتهى . ان ضغط زر الـ Play عندئذ يعيد بدء الوسائط من بدايتها .

- **DeviceType:** استخدم هذه الخاصية لتحدد أى نوع من الجهاز أو الملف يتحكم فيه الـ MediaPlayer . ان القيمة الافتراضية ، وهى dtAutoSelect ، تحدد نوع الجهاز ، وبذلك نوع محرك الجهاز الذى يستخدم ، على اساس إمتداد اسم الملف الذى تم اختياره كما هو مسجل فى سجل الـ Windows . لتحديد نوع معين من الاجهزة ، يمكنك ان تحدد الـ DeviceType باحدى القيم التالية : dtAVIVideo ، dtCDAudio، dtDAT ، dtDigitalVideo ، dtMMMovie ، dtOther ، dtOverlay، dtScanner ، dtSequencer ، dtVCR ، dtVideodisc ، أو dtWaveAudio.

- **Display:** تعين اسم component مثل الـ Panel أو الـ Form لهذه الخاصية . ثم يتم ارسال المخرجات من الـ MediaPlayer لهذا الـcomponent . ولكن بعض انواع الوسائط تعرض دائماً فى نافذة منفصلة . كنتيجة لذلك ، يتم



تجـاهل خـاصـيـة الـ Display وسـائط الـ Animation ، الـ AVI Video (ذات صوت)، الـ Digital Video، الـ Overlay ، والـ VCR .

- **FileName:** يعين اسم ملف الوسـائط الذى يجب فـتـحـه . اذا كـانت الـAutoOpen مـحددة بـ True ، فإن التغييرات التى تحدث للـ FileName تؤدى الى استدعاء الـ Open method بصورة تلقائية .

- **Frames:** هذه الخاصية تحدد عدد الاطارات التى تم زيادتها فى الوسـائط عندما ضغط المستـخدم زر الـ Step أو الـ Back . فى العـادى تكون الـ Frames محددة، بعد فتح ملف الوسائط، بـ ١٠٪ من عدد الاطارات الموجودة فى الملف . قم بتغيير هذه القيمة بعد استدعاء الـ Open لتغيير الاطارات الزائدة .

- **Mode:** إنك تختبر هذه الخاصية بشكل نموذجى فى OnNotify التابع للـ MediaPlayer (تأكـد من ان تحـدد الـNotify بـ True حتى يتلقى الـ procedure اخطارات الوسـائط) . ان الـ Mode تساوى واحدة من القيم الواضحة mpNotReady ، mpStopped ، mpPlaying ، mpRecording، mpSeeking ، mpPaused ، أوmpOpen .

افكار للمستخدم الخبير

- اذا لم تظهر المخرجات الجرافيكية الخاصة بالـobject بالشكل المتوقع، تأكد من انك قمت بتعيين قيمة لون لخاصية الـ Pen.Color التابعة للـ Canvas . اذا استـمـرت صـورتك غـيـر ظاهرة، تأكـد من انك ترسم فى الـ Canvas الصحيح . انتبه الى عبارات الـ with ، والتى قد تجعل العبارات تشير الى خـاصـيـة Canvas فى الـ object مـثل الـ bitmap بدلاً من الـ form أو الـ PaintBox المرادة .

- بسبب التغيرات فى الحاسب المكتبى للـ Windows 95 ، فقط اصبح الرسم على ايقونة- لتحريك تطبيق مصغر ، مثلاً ، غير محبذ . ولكن ، يمكنك ان تفعل هذا مع تطبيقات الـWindows 3.1 بتعيين TIcon object لخاصية الـ Application.Icon . تنحـدر الـ TIcon class من الـ TGraphic– استخدم الـ TIcon بنفس الطريقة التى تستخدم بها الـ TBitmap .

- لحذف صورة مثل bitmap كبيرة مرتبطة بـ Image object، قم بتمرير nil للـ Assign method بخاصية الـ Picture. على سبيل المثال، تمسح العبارة التالية أى بيانات للـ Image1، وهو ما قد تفعله لتحفظ الذاكرة:

```
Image1.Picture.Assign(nil);
```

- ان استدعاء الـ Update method للـ form بعد الـ Invalidate يعد امراً اختيارياً. ولكن، يجب عليك دائماً ان تستدعى الـ Invalidate قبل استدعاء الـ Update. ان استدعاء الـ Update وحده لا يفعل شيئاً. واستدعاء الـ Invalidate (والـ Update اختيارياً) يولد ايضاً OnPaint event لأية PaintBox objects فى الـ form.
- لا تقم باستدعاء الـ Update أكثر من مرة، أو انك سوف تجعل برنامجك كسولاً. ان الـ Windows يوحد الاستدعاءات المتعددة للـ Invalidate ويبعث رسالة wm_Paint عندما يكون كل شئ هادئ على جبهة الرسالة. استخدم الـ Update لكى يحدث تحديث فورى للشاشة.
- ان تحديد مواصفات الـ Height والـ Width للـ TBitmap تنسخ الـbitmap الى الذاكرة. يمكنك ان تغير ابعاد الـTBitmap فى أى وقت، ولكن اذا كانت الصورة اصغر حجماً، فإن أى pixels خارج الابعاد الجديدة يعتبر أنه قد فقد الى الابد.

المشروعات التى يمكنك تجربتها

(١٣-١): حاول إدخال الـ Image، والـ Shape، والـ PaintBox فى الـ Panel لإنشاء toolbar جرافيكية. يمكنك استخدام هذه الـ objects مع الـ SpeedButtons واللوحات الفرعية العادية لتجهز Status lines و toolbar خاصة بتطبيق.

(١٣-٢): اكتب تطبيقاً يعرض ملف الـ Calendar.bmp الخاص بـ Delphi الموجود على دليل الـ C:\...\Images\Backgrnd أضف ايقونات سحب واسقاط يمكن للمستخدمين تحريكها فى محيط يوم- مثلاً، للإشارة الى عيد ميلاد أو موعد ما.

(١٣-٣): اكـتب Utility يعـرض كل ملفـات الايقـونات فى دليل الـ C:\Delphi\Images\Icons .

(١٣-٤): اكتب برنامجاً يعرض ملف bitmap. (ملحوظة. استخدم Off Screen bitmap).

(١٣-٥): متقدم: قم بتطوير procedures الصور المتحركة التى تعرض bitmaps مخزنة فى Objects للـ TStringList. (ملحوظة: انظر الباب الثامن لمعرفة وصف owner-draw controls ومشروع الـ GlyphLst على القرص المدمج).

(١٣-٦): اكتب utility تستخدم الـ Animate لعرض صور Windows متحركة معيارية.

(١٣-٧): باستخدام التطبيق VideoPlayer كمرشد لك، اكتب برنامج AudioPlayer يمكنك ان يدير ملفات الـ (WAV).

ملخص:

- توفر عدد من components خاصية الـCanvas التى يمكنك استخدامها للرسم. تعتبر الـ Canvas object مشتقة من الـ TCanvas class، والتى تضم الـ Windows Graphics Device Interface (GDI) وعناصر الـ handle لـ device context.
- لتشكيل مخرجات جرافيكية، عين قيم لخصائص الـ Pen والـ Brush التابعة للـCanvas. لرسم شكل معين، استدع methods فى الـ Canvas مثل الـ Rectangle والـ Ellipse .
- استخدام الـ PaintBox لتوفير Canvas وإضافة امكانات جرافيكية للـobject مثل الـ Panel. يمكنك ايضاً استخدام واحداً أو أكثر من الـ PaintBox objects لقصر المخرجات الجرافيكية على المنطقة المستطيلة المحددة فى الـ form.
- تعتبر الـ TGraphic هى الـ class السابقة للـ TBitmap، والـ TIcon، والـTMetafile. تحتوى الـ TPicture class على الـ TGraphic object. يمكن لبرنامجك ان ينشئ TPicture objects لتخزين صور

جرافيكية فى الذاكرة. أو، يمكنك إضافة Image على form، واستخدام خاصية الـ Picture التابعة له لنفس هذا الغرض. هذا ايضاً ينطبق على الـ controls المنحدرة من الـ TGraphic مثل الـ TJPEGImage .

- أن TPicture class والتى تتمتع بالـ file-handling methods مثل الـ LoadFromFile والـ SaveToFile، تحدد نوع صورة الجرافيك من امتداد اسم الملف لذلك، ويستطيع الـTGraphic object ان يقرأ ويكتب meta file، أو bitmap، أو ايقونة. ويمكنه ايضاً التعامل مع الـ controls المنحدرة من الـ class مثل الـ TJPEGImage وغيرها.
- استدع الـ Draw methods أو الـ StretchDraw للـCanvas لرسم أى TGraphic object (meta file، أو bitmap أو ايقونة). استخدم الـ Draw لرسم الجرافيك فى احجامه المحددة. استخدم الـ StretchDraw لإعادة تحديد حجم الصور لتتلائم داخل المستطيل المحدد.
- توفر الـ Titmap class خاصية الـ Canvas التى يمكنك استخدامها لرسم offscreen Bitmap. استدع الـ CopyRect أو الـ BrushCopy methods التابعة للـ PaintBox offScreen أو الـ form لرسم الـ offScreen Bitmap.
- قم بإنشاء Shape objects للسحب والاسقاط من خلال الـ OnMouseDown، والـ OnMouseMove، والـ OnMouseUp، كما يوضح تطبيق الـ DragMe الخاص بهذا الباب.
- يستطيع الـ Animate عرض AVI clips صامتة، إما محملة من ملف قرص، أو مختارة من احدى الـ Windows clips المعيارية.
- يعمل الـ MediaPlayer كـ controller لاجهزة الوسائط المتعددة المتنوعة. يوضح هذا الباب كيفية استخدام الـ MediaPlayer لإنشاء متصفح ملف صور AVI.

فى واجهة التطبيق الجرافيكية، للمستخدم تعتبر الطباعة وعرض الصور الجرافيكية موضوعات مرتبطة ببعضها البعض الى حد كبير. يشرح الباب القادم امكانات طباعة النص والصور الجرافيكية الخاصة بـ Delphi.

*****



الباب الرابع عشر
تطوير تطبيقات الطابعة

محتويات هذا الباب:

- Components
- طباعة Plain-text
- الـ TPrinter class
- طباعة الجرافيك

لطباعة النص والجرافيك، تطلب منك كثير من نظم التطوير أن تستخدم أوامر الـ "escape" القديمة لـ Windows والتى يمكنها أن تجعل حتى modules الطباعة البسيطة مثل ألم الأسنان. لحسن الحظ، إن الـ functions والـ procedures والـDelphi components المتعلقة بالطباعة قد أزالت هذه الصعوبة وجعلت تطوير تطبيق الطابعة، مع مرور الوقت، شيئاً غير مؤلم.

فى هذا الباب، سوف أشرح كيفية عرض dialog الإعداد للطابعة وكيفية استخدام الـ Printers unit الخاصة بـ Delphi لطباعة النص والجرافيك. سوف أقترح أيضاً تقنية لإنشاء أمر مراجعة الطباعة النهائية- وهى ميزة يجب أن توفرها كل تطبيقات الطابعات بحيث يتمكن المستخدمون من فحص المخرجات على الشاشة، بدلاً من إهدار الصفحات الإختيارية.

:Components

فيما يلى قائمة بالـ components المتعلقة بالطابعة فى Delphi :

- **PrintDialog:** استخدم هذا الـ components ليستخدمه المستخدمين عند بداية عملية الطبع . يتضمن الـ Windows dialog components معيارى يمكن المستخدمون من تحديد صفحات الخاصة للطبع ويوفر وصولاً لخيارات الاعداد للطابعة . الـ Palette : Dialogs .

- **PrinterSetupDialog:** استخدم هذا الـ components لعرض واحداً أو أكثر من dialog الاعداد المتوفرة بواسطة برامج تشغيل الطابعة المركبة . يجب ان تضم دائماً هذا الـ dialog فى تطبيقات الطابعة لتعطى المستخدمين method لتشكيل الطابعات أو الاختيار من بين اجهزة الطابعة المتعددة وابواب الإخراج . الـ Palette : Dialogs .

- **TPrinter:** هذا الـ components غير موجود على الـ VCL palette ، ولكنه معرف كـclass فى الـ Printers unit . من بين اعضاءها الآخرين، توفر الـTPrinter class خاصية Canvas والتى تعطيك سطح رسم WYSIWYG لطباعة النص والجرافيك الـ Palette : لا يوجد .

:Plain text

تستطيع تطبيقات Delphi ان تطبع النص بإحدى طريقتين . فى هذا الفصل ، سوف اشرح اسهل الطرق باستخدام الـ Write procedures والـ Writeln (تنطق write-line) التابعة للـ Pascal مع ملف إخراج الـ Text . ان هذا لا يعد ملف قرص ، ولكن ملف بالمعنى العام للبرمجة للتمييز عن فيض البيانات . عندما يكون لديك نصاً فقط للطباعة ، فهذه هى التقنيات التى يجب ان تستخدمها .

Note

ملحوظة: ان تقنيات الـ Write والـ Writeln التالية تطبع النص باستخدام الـ TrueType Font الجرافيك الاخرى ، والذى يمكنك ان تضعه بنمط سميك أو مائل اذا اردت . حتى هذه التقنيات مخصصة لطباعة النص بشكل صارم ، إن هذه الطرق لا تستخدم مجموعة الرموز الافتراضية للطباعة الا اذا قمت بتركيب برنامج تشغيل طابعة نص فقط . ان بعض الطابعات الحديثة تخطط الـ TrueType Font مباشرة لـ Font الطابعة الداخلية ، ولكن لكل الاغراض العملية ، يعتبر هذا الفعل شفافاً .



الـ Printers unit:

لكل module تحتاج الى إمكانيات طباعة ، أضف الـ Printers unit لتعريف الـ uses الخاص بالـ module . على سبيل المثال ، ابدأ مشروعاً جديداً وقم بتعديل تعريف الـ uses الخاص بالـ unit الرئيسية مثل هذا (الإضافة الجديدة هى المكتوبة بخط سميك) :

```
uses
   Windows, Messages, SysUtils, Classes, Graphics,
 Controls,
   Forms, Dialogs, Printers;
```

الـ Printers unit توفر الـ Printer object والتابع للـ TPrinter class . ان كل اوامر اخراج الطابعة تمر على هذا الـ object ، والذى يوفر ايضاً معلومات مفيدة مثل عرض وطول الصفحة بالـ pixels .

Tip

فكرة: قد يكون هذا واضحاً ، ولكن يجب الإشارة الى أن الـ Printers unit هى جمع ؛ بينما الـ Printer object مفرد . اذا واجهت مشكلة فى بدء الطباعة ، تأكد من أنك لم تخلط بين الكلمتين .

سوف أصف كل عضو فى الـ TPrinter class مؤخراً فى هذا الفصل . ولكن أولاً ، لنلقى نظرة على عملية طبع بسيطة توضح كيفية استخدام الـ Printers unit والـ Printer object التابع لها . إوصل طابعتك ، وقم بتشغيلها ، وإتبع هذه الخطوات :

١- ابدأ تطبيقاً جديداً . إنتقل لنافذة للـ code editor ، وفى القمة عند تعريف الـ uses ، أضف Printers الى قائمة الـ units الأخرى التى تستخدمها هذه الـ module .

٢- أضف Button object من لوحة الـ Standard على الـ form واضغط الـ Button مرتين لإنشاء الـ OnClick .

٣- انسخ البرمجة الموجودة فى القائمة (١٤-١) الى الـ Button1Click procedure . توضح الـ code تقنيات طباعة نص أولية .

٤- قم بتشغيل البرنامج بضغط F9 . عندما تظهر نافذة البرنامج ، اضغط الزر لطباعة الـ string التالى فى أعلى الصفحة (إعتماداً على نوع الطباعة التى لديك ، قد تكون المخرجات صغيرة جداً) :

```
Hello printer!
```

القائمة (١٤-١): OnClick الخاص بهذا الـ Button يظهر تقنيات طباعة نص أولية

```
procedure TForm1.Button1Click(Sender: TObject);
var
  FPrn: System.Text;
begin
    AssignPrn(FPrn);
    Rewrite(FPrn);
    try
        Writeln(FPrn, 'Hello printer!');
      finally
        CloseFile(FPrn);
      end;
end;
```

لتوفير ملف يبعث إليه البرنامج مخرجات النص procedure، يعرف الـ FPrn (يمكنك أن تسميه اسماً آخر) من نوع الـ System.Text. قم بأداء خطوتين لإلحاق متغير الملف إلى procedures الإخراج الخاصة بالـ Pascal وقم بفتحه للعمل:

```
AssignPrn(FPrn);  { Attach file for output }
Rewrite(FPrn);    { Open the file }
```

الـ Printers unit توفر الـ AssignPrn لتعيين ملف المخرجات ولتوفير Buffer للمخرجات بالذاكرة. الـ Rewrite ويعتبر Pascal procedure معيارى يفتح ملف نص. بعد هاتين الخطوتين، يمكنك استدعاء الـ Writeln لطباعة strings. (إن اولائك الذين يعرفون الـ Pascal ولكنهم تطوروا سريعاً الى الـ Windows قد لا يعرفوا الـ write والـ Writeln، لذا سأسترجعهم سريعاً هنا). إن استدعاء الـ Writeln يطبع string ويبدأ خطاً جديداً لتولى أية exception محتملة، استخدم الـ FPrn داخل الـ try:

```
try
  Writeln(FPrn, 'Plain text is plain and simple!');
    { ... insert other output statements here }
```

أغلق الطابعة، التى تنهى مهمة الطبع وأخرج أى صفحة تامة بشكل جزئى، باستدعاء الـ CloseFile لتكون فى أمان، أدخل هذه العبارة فى الـ finally، والذى يعتبر مضمون فى أن ينفذ حتى اذا فشل الطبع :

```
finally
  CloseFile(FPrn);
end;
```

لاحظ أن الـ end مطلوبة لإنهاء عبارة الـ try-finally، رغم أنه لا يوجد begin . يمكنك أيضاً استخدام الـ Writeln لطباعة قيم متعددة، اعداد صحيحة، متغيرات نقطة عائمة، objects أنواع البيانات البسيطة الأخرى :

```
Writeln(FPrn, 'Number of components = ', ComponentCount);
```

استخدم Write لطباعة النص بدون أن تبدأ سطر جديد . على سبيل المثال، السطور التالية مساوية للعبارة الفورية السابقة :

```
Write(FPrn, 'Number of components = ');
Write(FPrn, ComponentCount);
Writeln(FPrn);  { Start new line }
```

ان محاولة تمرير ملف مخرجات طابعة والذى يكون قد تم تعينه بـ AssignPrn للـ Read أو الـ Readln يولد Run-time error exception . لايمكنك قراءة البيانات من على جهاز مخرجات الطابعة .

الـ Fonts والـ Control codes:

ان الـ Write والـ Writeln يستجيبان لاربعة control codes، وهى مذكورة فى جدول (١٤-١)، والتى يمكنك استخدامها لارسال اوامر للطابعة .

جدول (١٤-١): control codes الاربع الخاصة بالـ Write والـ Writeln

control codes	الأمر
#9	Tab
#10	New Line
#13	Carriage Return
^L	New Page

يمكنك ضم قيم الـ ASCII هذه strings أو طباعتها بعبارات Write مثل هذه :

```
Write(FPrn, #9);   { Tab }
Write(FPrn, #13);  { Flush output buffer }
Write(FPrn, #10);  { Flush and start new line }
Write(FPrn, ^L);   { Flush and start new page }
```

تقوم الـ Write والـ WriteIn بتحديد الـ tabs الى ثمانية أضعاف متوسط عرض الرمز بالـ pixels للـ Font الحالى . عند الطباعة بـ font نسبى ، هذا يعنى أنك لا تستطيع إستخدام الـ tabs لمحاذاة الأعمدة الغير رقمية بالـ Write والـ WriteIn . فى معظم الـ fonts النسبية ، رغم ذلك ، تكون الأرقام أحادية المسافة من حيث الحجم ، لذا قد تريد إستخدام code لضبط الـ tab لمحاذاة الأعمدة الرقمية الخالصة .

يوفر الـ Printer خاصية الـ Canvas التى ، من بين مهارتها الأخرى ، تحدد font الطابعة ، بـ ١٠ نقاط حسب النظام الإفتراضى يمكنك تغيير font الطابعة بتعيين أسمه لخاصية الـ Font.Name التابعة للـ Canvas . على سبيل المثال ، إستخدم هذه العبارات لإختيار الـ TrueType Courier New font بـ ١٢ نقطة لعبارات الـ Write والـ WriteIn المتالية :

```
with Printer.Canvas do
begin
  Font.Name := 'Courier New';
Font.Size := 12;
end;
```

يمكنك إضافة هذا الـ code للمثال السابق فى Button1Click . إدخل السطور الخمسة المذكورة هنا بعد عبارة الـ Rewrite مباشرة (فوق الكلمة الأساسية try) . والآن عندما تضغط F9 لتشغيل البرنامج وتضغط الزر ، يتم طباعة النص بـ ١٢ نقطة أحادية المسافة .



تحذير: عند طباعة نص عادى بإستخدام التقنيات الموضحة فى هذا الفصل ، إستدع دائماً الـ AssignPrn والـ Rewrite قبل تغيير قيم خاصية الـ Font فى الـ Canvas التابع للـ Printer .

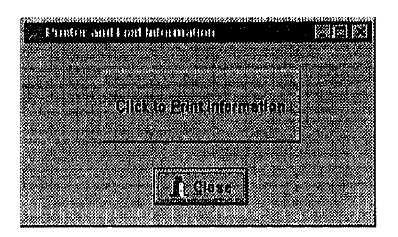
يؤثر الـ font المختار على عرض وطول character pixels ، ولذلك ، بتغيير عـدد السطور في كل صفحة يمكنك إستخدام أى font مـركب ، ولكن الـ TrueType تعتبر الأفضل دائماً للطباعة لأن الـ Windows GDI يمكن أن يقدم character لأى driver ، حتى لو لم يكن يستخدم الـ TrueType مباشرة .

يجب أن تعرف أيضاً كيف تستخدم تقنيات الـ Write والـ WriteIn الموضحة هنا للطباعة على طابعات النص فقط مثل تلك الأنواع القديمة ، إذا كان يجب عليك التعامل مع طابعات النص فقط ، إستخدم Control Panel لتركيب برنامج التشغيل الـ Windows Generic/Text Only وتشغيله . ولكن لا يوجد ضمان أن هذا سوف يعمل ، وللحصول على أفضل النتائج ، يجب أن تحدد للمستخدم النهائى لبرنامجك أن طابعة الجرافيك مطلوبة للطباعة مع الـ Windows .

إحصائيات الطابعة

On the CD-ROM

على القرص المدمج : إن تطبيق الـ PrnInfo الموضح فى شكل ١٤-١ يطبع إحصائيات عن إمكانيات طابعتك والـ font الحالى . توجد ملفات البرنامج على القرص المدمج فى دليل الـ Source\PrnInfo . يوضح البرنامج أيضاً كيفية تحديد عدد السطور لكل صفحة ، والذى قد يكون مفيداً لبعض التطبيقات .



شكل (١٤-١): إضغط الزر الكبير لـ PrnInfo للحصول على تقرير مطبوع عن برنامج تشغيل طابعتك، وعدد السطور فى كل صفحة، ومتوسط عدد الرموز لكل سطر.

لإستخدام البرنامج ، حمل ملف مشروع الـ PrnInfo فى الـ Delphi ، إفتح طابعتك ، إضغط F9 لتشغيل البرنامج ، وإضغط الزر الكبير . وما قام الـ PrnInfo بطباعته هو عن مواصفات نظامى .

```
Device = HP DeskJet 720C Series on LPT1:
Font = Courier New
Font Size = 12 points
PageHeight = 3150 pixels
PageWidth = 2400 pixels
Extent.Cx = 6720 pixels
Extent.Cy = 53 pixels
Lines per page = 56
Chars per line = 80
```

توضح القائمة (١٤-٢) source code للـ PrnInfo.

القائمة (١٤-٢): PrnInfo\Main.pas.

```
unit Main;

interface

uses
  SysUtils, Windows, Messages, Classes, Graphics,
    Controls, Forms, Dialogs, Printers, StdCtrls, Buttons;

type
  TMainForm = class(TForm)
      PrintButton: TButton;
        CloseBitBtn: TBitBtn;
        procedure PrintButtonClick(Sender: TObject);
       private
   { Private declarations }
       public
{ Public declarations }
       end;

var
  MainForm: TMainForm;
```

```
implementation

{$R *.DFM}

procedure TMainForm.PrintButtonClick(Sender: TObject);
var
 FPrn: System.Text;
   Extent: TSize;
   Metrics: TTextMetric;
   I, LinesPerPage, CharsPerLine, AverageWidth: Integer;
   S : String;
begin
 AssignPrn(FPrn);
   Rewrite(FPrn);

   with Printer.Canvas do
   begin
    Font.Name := 'Courier New';
      Font.Size := 12;
     end;

{ Fill test string with ASCII values 32 to 255 }
 try
(*   S[0] := Chr(224);     // This is no longer allowed *)
    SetLength(S, 224);      // Use this method instead
    for I := 32 to 255 do   // Fill string with test chars
     S[I - 31] := Chr(I);
      with Printer, Canvas do
    begin
    { Determine number of lines per page }
     GetTextExtentPoint(Handle, @S[1], Length(S),
   Extent);
      LinesPerPage := PageHeight div (Extent.Cy + 2);
```



```
    if PageHeight mod Extent.Cy <> 0 then
        Dec(LinesPerPage);
        { Determine average number of characters per line }
        GetTextMetrics(Handle, Metrics);
        AverageWidth := Metrics.tmAveCharWidth;
        CharsPerLine := PageWidth div AverageWidth;
  { Print the report }
       Writeln(FPrn, 'Device = ', Printers[PrinterIndex]);
        Writeln(FPrn, 'Font = ', Font.Name);
        Writeln(FPrn, 'Font Size = ', Font.Size, ' points');
        Writeln(FPrn, 'PageHeight = ', PageHeight, ' pixels');
        Writeln(FPrn, 'PageWidth = ', PageWidth, ' pixels');
        Writeln(FPrn, 'Extent.Cx = ', Extent.Cx, ' pixels');
        Writeln(FPrn, 'Extent.Cy = ', Extent.Cy, ' pixels');
        Writeln(FPrn, 'Lines per page = ', LinesPerPage);
        Writeln(FPrn, 'Chars per line = ', CharsPerLine);
  end;
 finally
    CloseFile(FPrn);
     end;
  end;

  end.
```

فى المعلومات المذكورة، يساوى الـ Extent.Cx عدد الـ pixels الأفقية التى ستكون مطلوبة لطباعة الـ string الإختبارى للبرامج بإستخدام الـ font الحالى (حدد الـ Courier New بـ ١٢ نقطة). إن الـ PrnInfo لا يطبع هذا الـ string - فهو يستخدم فقط لإعداد التقرير. ويساوى الـ Extent.Cy الإرتفاع بالـ pixels للـ string الإختبارى. ويكون عدد السطور لكل صفحة دقيق فقط إذا قمت بطباعة كل السطور على الصفحة بنفس الـ font. وعدد الرموز فى كل سطر يكون تقريبى للـ pixels النسبية.

وتكون هذه القيمة عادة حاسمة فى تحديد كم المعلومات التى تتناسب داخل صفحة من المخرجات. تأكد من إختيار اسم font وحجمه، كما هو موضح فى القائمة، قبل الحصول على المعلومات.

لعرض إحصائيات عن font مختلف ، أضف عبارات مثل التالية قبل إستدعاء الـ GetTextExtentPoin مباشرة :

```
Font.Name := 'Arial';
Font.Size := 24;
```

يوضح الـ PrnInfo كيفية إستدعاء الـ Windows API functions التى تتطلب handle للـ printer device قم بتمرير خاصية الـ Handle الخاصة بالـ Canvas للـ Printer إلى هذا الـ parameter . على سبيل المثال ، هذا يؤدى إلى إستدعاء GetTextExtentPoint :

```
GetTextExtentPoint(Handle, @S[1], Length(S), Extent);
```

لإتمام الـ function PChar parmeter ، تمرر العبارة عنوان الرمز الأول (@S[1]) للـ string الإختبارى . وتستدعى عبارة مشابهة API function أخرى ، وهى GetTextMetrics :

```
GetTextMetrics(Handle, Metrics);
```

إستخدم هذه التقنيات وقيم الـ PrnInfo الواردة ، فقط مع method طباعة النص الخاصة بالـ Write والـ WriteIn الموضحة هنا . كما سأشرح فيما بعد فى هذا الباب ، فإن تقنيات الطباعة الأخرى تعطيك حكماً أفضل على النتائج ، خاصة عند طباعة الجرافيك .

طباعة قوائم الـ string

لطباعة سطور نص خاصة بطباعة الـ TStrings أو الـ TStringList ، إستخدم الـ Writeln كما هو موضح فى هذا الفصل . أضف الـ Printers لأمر الـ uses بالـ module وإدخل هذا الـ code فى Button OnClick لطباعة قائمة string بإستخدام menu :

```
var
  FPrn: TextFile;
    I: Integer;
begin
  AssignPrn(FPrn);
```

```
Rewrite(FPrn);
  try
    for I := 0 to Memo1.Lines.Count - 1 do
        Writeln(FPrn, Memo1.Lines[I]);
        finally
    CloseFile(FPrn);
      end;
end;
```

يمكنك إستخدام هذا الـ code لطباعة أى خاصية TStrings - على سبيل المثال كما هو موضح هنا أو خاصية الـ Items فى الـ ListBox .

dialogs الطابعة

إن methods واجهة التطبيق للمستخدم للطباعة تختلف فيما بين التطبيقات، ولكن أغلب البرامج تستخدم على الأرجح واحداً من أوامر قائمة الملف التقليدية . وأوامر القائمة المعيارية الأربعة هى :

* **File|Page Setup** - اختيارى . يعرض Dialog تطبيق والذى ينظم المسافات الرأسية، وأرقام الصفحات، والهوامش، والتحديدات الأخرى المتنوعة التى تحتاجها . لايوجد أية Methods معيارية لتنفيذ هذا الأمر، غير أنه يجب أن يسمح للمستخدمين بإضافة مواصفات وأية بنود تكون فريدة فى مميزات الطباعة الخاصة بتطبيقك . على سبيل المثال، يمكنك أن تحث المستخدمين على إدخال string ليتم طباعته فى أعلى كل صفحة .

* **File|Print Preview** - إختيارى يعرض صفحات مطبوعة مقلدة فى النافذة . لا يوجد أية طرق معيارية لتنفيذ هذا الأمر، ولكن مؤخراً فى هذا الباب سوف أقترح تقنية سوف تعمل لجميع التطبيقات .

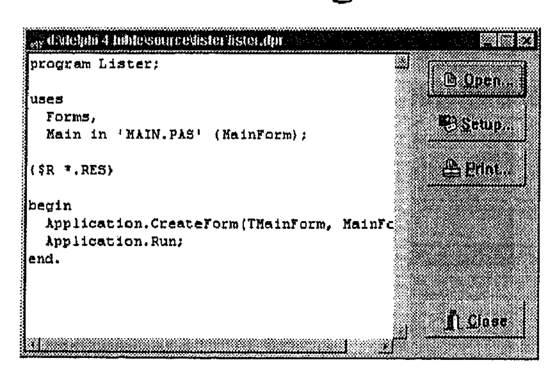
* **File|Print Setup** - يعرض dialog الإعداد لبرنامج تشغيل الطابعة الحالى، والذى يتضمن زر Options يفتح dialog إضافية . هذا الأمر يجب ألا يطبع أى شئ . إستخدم الـ PrinterSetupDialog لتبسيط تنفيذ هذا الأمر .

* **File|Print** - يعرض الـ dialog مواصفات الطابعه للمستخدمين لكى يدخلوا عدد النسخ التى يريدونها ومتوسط عدد الصفحات . إذا ضغط المستخدم زر الـ OK



لـ dialog ، يجب أن يبدأ برنامج الطباعة على الفور . والـ dialog box الخاص بهذا الأمر يشمل زر Setup الذى يعرض نفس dialog الإعداد مثل أمر الـ File|Print Setup . إستخدم الـ PrinterDialog لتبسيط تنفيذ هذا الإمر .

على القرص المدمج : يعتبر أول أمرين إختياريين . ويعتبر الـ File|Print Setup، والـ File|Print معياريين ، بالرغم من أنه بإستطاعتك أن تستمر مع الـ ...File|Print فقط لأن هذا الإمر يتيح الوصول إلى dialog إعداد برنامج التشغيل ولكن معظم التطبيقات يجب أن تنفذ على الأقل أمرى الـ File|Print Setup والـ ...File|Print يوضح تطبيق الـ Lister على القرص المدمج - والموضح فى شكل (١٤-٢) والقائمة (١٤-٣) - كيفية إستخدام الـ PrintDialog والـ PrinterSetupDialog التابعة لـ Delphi فى كتابة أوامر الـ ...File|Print والـ ...File|Print Setup الخاصة بالبرنامج . ويستطيع برنامج الـ Lister أن يطبع أى ملف نص pas. أو txt. ، أو أى ملف نص آخر . لعرض النص والبيانات الرقمية الخاصة بالطباعة ، فإن الـ Lister يسبق السطور بأرقام للسطر . ويستخدم النص المطبوع الـ Courier New font ذى العشر نقاط . توجد كل ملفات الـ Lister على القرص المدمج فى دليل الـ Source\Lister .



شكل (١٤-٩٢)يطبع تطبيق الـ Lister أى ملف نص ويعرض كيفية تنفيذ أمرى الـ File|Print والـ File|Print Setup.

القائمة ١٤-٣: Lister\Main.pas

```
unit Main;

interface
```

```
uses
 SysUtils, Windows, Messages, Classes, Graphics,
    Controls, Forms, Dialogs, StdCtrls, Buttons, Printers;

type
    TMainForm = class(TForm)
      Memo1: TMemo;
      SetupBitBtn: TBitBtn;
      PrintBitBtn: TBitBtn;
      CloseBitBtn: TBitBtn;
      OpenBitBtn: TBitBtn;
      OpenDialog1: TOpenDialog;
      PrintDialog1: TPrintDialog;
      PrinterSetupDialog1: TPrinterSetupDialog;
      procedure OpenBitBtnClick(Sender: TObject);
      procedure SetupBitBtnClick(Sender: TObject);
      procedure PrintBitBtnClick(Sender: TObject);
     private
      { Private declarations }
     public
      { Public declarations }
     end;

var
 MainForm: TMainForm;

implementation

{$R *.DFM}

procedure TMainForm.OpenBitBtnClick(Sender: TObject);
begin
 with OpenDialog1 do
    if Execute then
    begin
```

```
        Memo1.Lines.LoadFromFile(FileName);
        Caption := Lowercase(FileName);
      end;
end;

procedure TMainForm.SetupBitBtnClick(Sender: TObject);
begin
    PrinterSetupDialog1.Execute;
end;

procedure TMainForm.PrintBitBtnClick(Sender: TObject);
var
  FPrn: System.Text;   { Printer output text file }
    I: Integer;        { Memo1.Lines index }
    LCol: Integer;     { Line number column width }
begin
  if PrintDialog1.Execute then
    begin
      AssignPrn(FPrn);  { Direct Write/Writeln to FPrn }
      Rewrite(FPrn);    { Open printer output file }
      try
       Printer.Canvas.Font := Memo1.Font;  { Use Memo's font }
       with Memo1, Lines do
       begin  {Set line number column width for file size:}
        if Count < 10 then LCol := 1        {0 .. 9 lines    }
        else if Count < 100 then LCol := 2  {10 .. 99 lines  }
        else if Count < 1000 then LCol := 3 {100 .. 999 lines}
        else LCol := 4;    { You must be kidding }
        for I := 0 to Count - 1 do
        begin
         Write(FPrn, I + 1:LCol, ': ');  { Print line number }
          Writeln(FPrn, Lines[I]);       { Print line }
        end;
       end;
      finally
       CloseFile(FPrn);  { Close printer output file }
```



```
        end;
      end;
  end;

  end.
```

لتنفيذ أمر الـ File|Print Setup...، كما فى برنامج الـ Lister، أضف الـ PrinterSetupDialog على الـ form. يتم تمثيل الـ object بأيقونة، والتى لا تظهر فى وقت التشغيل. وداخل OnClick للـ Button أو لأمر قائمة، إدخل هذه العبارة:

```
     PrinterSetupDialog1.Execute;
```

أو، يمكنك القيام بفعل آخر إذا أغلق المستخدمون dialog الإعداد بإختيار زر الـ OK له، بالرغم أن هذا ليس ضرورياً:

```
  if PrinterSetupDialog1.Execute then
    { ... take action if user selected OK }
```

قم بتنفيذ أمر الـ File|Print بأسلوب مشابه، ولكن أضف PrintDialog (وهذا أيضاً ليس له ظهور فى وقت التشغيل).

توضح القائمة (٤-١٤) التخطيط العام الـ OnClick الخاص بالأمر. وداخل الـ try، قم بإضافة عبارات Write أو الـ Writeln كما يوضح procedure الـ PrintBitBtnClick لتطبيق الـ Lister.

القائمة (٤-١٤): نفذ أمر الـ File|Print بإستخدام هذا النوع من الـ code.

```
  if printDialog1. Execute then
  begin
    AssignPrn(FPrn);
      Rewrite(FPrn);
      try
        { ... Write text to FPrn }
      finally
        CloseFile(FPrn);
      end;
  end;
```

فى أغلب الحالات ، إنك تستخدم نفس الـ font للطباعة الذى تستخدمه لعرض النص على الشاشة ، عند إستخدام component مثل الـ Memo لعرض نص ، عين الـ Font الخاص بالـ object للـ Canvas الخاصة بالـ Printer بعبارة مثل التالية ، ولكن تأكد من فتح ملف المخرجات قبل هذا :

```
Printer.Canvas.Font := Memo1.Font;
```

وإستخدم أية معلومات متوفرة فى الـ PrintDialog أمر يرجع إليك . بعد إستدعاء الـ Execute ، إستخدم قيم خصائص الـ Copies ، الـ FromPage ، الـ ToPage ، وقيم خصائص أخرى فى الـ print loop الخاصة بك :

```
with printDialog1 do
for Copy := 1 to Copies do
  for Page := FromPage to ToPage do
     { ... Print pages here }
```

الـ TPrinter class:

قبل الإنتقال للفصل التالى ، والذى يشرح كيفية طباعة الجرافيك ونص الـ WYSIWYG ، إقرأ التحليلات التالية لخصائص الـ TPrinter methods إنك تحتاج لفهم جيد لهذه الأشياء عندما تحتاج عملية الطباعة الخاصة بك أن تمتد إلى مابعد الـ Write procedures والـ WriteLn الخاصة بـ Delphi استخدم الـ Printer object للإشارة إلى كل methods والخصائص المذكورة فى هذا الفصل . على سبيل المثال ، التعبير Printer.Canvas يؤدى إلى الوصول إلى الـ Canvas الخاص بالـ Printer لإستخدام هذه الخصائص والـ method ، أضف Printers إلى أمر الـ uses للـ module الخاصة بتطبيقك .

خصائص الـ TPrinter

فيما يلى قائمة بخصائص الـ TPrinter : -

• **Aborted** - هذه الخاصية تصبح True إذا ما قام المستخدم بإنهاء (عمل) الطباعة . مثلاً بإغلاق الـ Print Manager . إفحص متغير الـ Boolean هذا فى الـ print loop الخاصة بك لتحديد ما إذا كان الـ TPrinter Abort method قد تم إستدعاءه ، وفى هذه الحالة ، يجب أن تنهى الـ loop فوراً دون إستدعاء الـ EndDoc .

- **Canvas** - لطباعة الجرافيك ونص الـ WYSIWYG، عين قيم خصائص الـ Pen والـ Brush والخصائص الأخرى للـ Canvas وإستدع method الخاص بها، تماماً مثلما تفعل لعرض الجرافيك.
- **Capabilities** - هذه الخاصية تذكر مجموعة إمكانيات برنامج تشغيل جهاز الطابعة. إنها مجموعة من قيم الـ pcCopies، والـ pcOrientation، والـ pcCollation.
- **Copies** - تساوى عدد النسخ التى تم طبعها.
- **Fonts** - تعرض قائمة TStrings بكل الـ fonts التى توفرها الطابعة الحالية. يمكنك دائماً الطباعة بإستخدام الـ TrueType fonts. ولكن هذه لن تكون مشكلة للـ Windows 95, 98, NT.
- **Handle** - قم بتمرير هذه القيمة لأى Windows API function تتطلب device context (HDC) handle على سبيل المثال، الـ GetText Metrics.
- **Orientation** - لهذه الخاصية إحدى قيمتين: الـ poPortrait أو poLandsCape. يمكنك تعيين هذه القيم قبل الطباعة، أو يمكنك فحصها لتحديد parameters الصفحة.
- **PageHeight** - يساوى طول صفحة طابعة واحدة بالـ pixels. وهذه القيمة يمكن أن تتغير بشكل كبير فيما بين الطابعات.
- **PageNumber** - تساوى رقم الصفحة الحالية. ومعنى الصفحة يرجع إلى تطبيقك. فإن الـ Printer يزيد الـ PageNumber فى كل مرة تستدعى فيها الـ NewPage method. لطباعة النص العادى، يتم زيادة الـ PageNumber عندما يبدأ الـ Writeln صفحة جديدة.
- **PageWidth** - يساوى عرض صفحة طابعة واحدة بالـ pixels. هذه القيمة قد تختلف كثيراً بين الطابعات.
- **PrinterIndex** - يساوى Index اسم الطابعة الحالية فى قائمة الـ string Printers.
- **Printing** - يساوى True أثناء عملية الطبع.



• **Title** – يحتوى string ، والذى يمكنك تعيينه ، ليحدد مهمة الطبع فى الـ Print Manager أو للشبكه .

الـ TPrinter methods

فيما يلى قائمة بالـ TPrinter methods : -

• **Abort** – يمكنك إستدعاء الـ Abort لإنهاء عملية الطباعة العاملة . على سبيل المثال ، يمكنك عرض modeless dialog box قبل أن يقوم تطبيقك بإضافة الـ print loop الخاصة بك . إستخدم flag عام لتشير إذا ما كان كذلك ، إستدع الـ Printer. ABORT وإخرج من الـ printer-output loop (لا تستدع أيضاً EndDoc) إن إستدعاء الـ Abort يحدد الـ Printer.Aborted flag بـ Ture .

• **BeginDoc** – إستدع هذا قبل بدء مهمة طباعة جديدة . لا يجب عليك إستدعاء الـ BeginDoc عند إستخدام تقنيات النص العادى للـ Write والـ Writeln الموضحة فى بداية هذا الباب .

• **EndDoc** – إستدع هذا بعد إنهاء عملية الطباعة . إن الـ EndDoc يفرغ Buffer المخرجات ويخرج الصفحة الأخيرة إذا لزم الأمر . لا تستدع الـ EndDoc عند إستخدام تقنيات الـ plain-text للـ Write والـ Writeln . كذلك ، لا تستدع الـ EndDoc بعد إنهاء عملية الطبع بإستدعاء الـ Printer.Abort (لكن هذا ليس خطيراً فى النسخ 3 و4 لـ Delphi كما كان فى Delphi 1& 2) .

• NewPage – إستدع هذا الـ procedure فى أى وقت لبدء صفحة جديدة . إن معنى ومحتوى الصفحة يرجع إليك وعلى برنامجك أن يوفره . إن الـ NewPage يقوم بتجميع Buffer المخرجات ودفع الصفحة الحالية .



تحـذير: لا تقم بإنشاء object من الـ TPrinter class . إن الـ unit Printer توفر الـ Printer object من نوع بيانات هذه الـ class ليكون معداً للإستخدام .

طباعة الجرافيك

إن تقنيات طباعة النص الموضحة فى الفصل السابق تمثل منطقة وسط بين سهولة الإستخدام والنتائج . عندما تحتاج إلى مزيداً من التحكم فى parameters

المخرجات - لتحصل على نص موضوع بدقة على الصفحة، مثلاً، أو للتعامل مع الـ tabs للحصول على fonts نسبية - يمكنك إستخدام الـ method التالية. في هذا الفصل، سوف أوضح كيفية طباعة الـ forms، الجرافيك، نص الـ WYSIWYG، والـ bitmaps والتطبيق على الـ font، الخاص بهذا الفصل، وهو الـ FontSnap، يقدم أيضاً تقنيات ذات هدف عام لمعاينة شاشة المخرجات المطبوعة.

طباعة الـ forms:

قد تريد أن تتضمن أمراً يطبع لقطات من مظهر الـ form على الشاشة. هذا سهل - فقط إستدع الـform print method. لا يجب عليك أن تستدع أي من الـ TPrinter method، أو إضافة Printers لتعريف الـ uses الخاص بالـ module.

لطباعة form، أضف PrintDialog على الـ form (هذا لا يظهر في وقت التشغيل أو على المخرجات المطبوعة)، ثم قم بتنفيذ OnClick للـ Button أو لأمر القائمة بعبارة مثل هذه:

```
if PrintDialog1.Execute then
  Print;
```

لطباعة form ما بالأسم، إستخدم هذه العبارة:

```
AboutBox.Print;
```

والـ AboutBox هو خاصية الـ Name التي تعينها للـ form. داخلياً، يقوم الـ form Print method بنسخ الـ client area الخاصة بالنافذة الحاليَة إلى bitmap بعيدة عن الشاشة. ثم يقوم الـprint method بعد ذلك بطباعة هذه الـ bitmap بإستدعاء الـ Windows API StretchDIBits function. قم بتعين واحدة من ثلاث قيم لخاصية الـ form PrintScale لتحديد كيف يتم قياس المخرجات. يمكنك أن تفعل هذا في وقت التشغيل أو في الـ Object Inspector.

- **poNone-No**. إن حجم الـ form المطبوعة يعتمد على resolution الطباعة. إن إستخدام هذا الخيار لطباعة forms على طابعات الليزر ينتج عنه نوافذ صغيرة تكون حادة ولكن غير مقروءة. إنك لن تستخدم هذا الخيار غالباً.
- **poPrintToFit** - يحدد حجم الصفحة. تملأ الـ form المطبوعة الصفحة في إتجاه واحد على الأقل (أفقياً غالباً). هذا الخيار يعتبر أكثر الخيارات إهداراً لحبر



الطباعة ، ولكن عندما تزيد للـ form أن تملأ الصفحة ، وهذا هو الخيار الصحيح للإستخدام .

- **poProportional** - يحدد بإستخدام خاصية الـ pixels فى كل بوصة (أو الـ pixels-per-inch) للطباعة . هذا يؤدى إلى مخرجات ذات أفضل شكل ، والتى تبدو ، من مسافة الرؤية العادية ، شبيهة فى الحجم لمظهر الـ form على الشاشة . هذا لا يعنى أن الأحجام تتماشى بشكل دقيق ؛ إن المخرجات المطبوعة تعتبر منطقة وسط مقبولة بين القراءة والمسافة .

طباعة object الجرافيك

إن تقنيات طباعة الجرافيك ونص الـ WYSIWYG . وهو النص المطبوع كـ object الجرافيك - تتطلب تقنيات مختلفة عن طباعة النص العادى بعبارات الـ Write والـ Writeln . إنك تستخدم نفس الـ Printers unit والـ Printer object ، ولكنك تستدعى TPrinter class method لبدء وإنهاء عملية الطباعة . يجب عليك تعريف معنى الصفحة لتطبيق ورسم كل output object هذا يتطلب مزيداً من العمل ، ولكنه يوفر أعلى سيطرة على النتائج .

On the CD-ROM **على القرص المدمج :** توضح القائمة ١٤-٥ الخطوات الأساسية المطلوبة لطباعة محتويات الـ Image . (يوجد هذا النص فى ملف Print1.pas على القرص المدمج فى دليل الـ (Source\PrintMisc) .

لا يجب عليك إستخدام الـ component ، ولكنه أفضل لهذا البرنامج . والخطوات الثلاث الأساسية هى : -

١ - إستدع Printer BeginDoc procedure لبدء الطباعة .

٢- إستدع Canvas method الخاص بالـ Printer .

على سبيل المثال ، إستدع Draw لطباعة TGraphic ، bitmap أو أيقونة أو meta file) ، أو يمكنك إستدعاء methods مثل الـ Ellipse والـ Rectangle . للحصول على تحكم أفضل للخطأ ، قم بهذه الخطوات داخل الـ try .

٣- إستدع الـ EndDoc عندما تنتهى من الطباعة . للحصول على تحكم أفضل للخطأ ، إستدع الـ EndDoc فى الـ finally .

القائمة (١٤-٥): اتبع هذه الخطة العامة لطباعة الجرافيك.

```
proceducre TForm1. printClick (Sender: TObject);
begin
  Printer.BeginDoc;
    try
      Printer.Canvas.Draw(0, 0, Image1.Picture.Graphic);
        finally
      Printer.EndDoc;
        end;
end;
```

لتـجـربة الـ procedure الموجـود فى القـائمـة ١٤-٥ ، أضف Image وButton على الـ form . إن حجم الـ Image غير مهم .

إستخدم خاصية الـ Image Picture لتحميل أى ملف . إضغط bitmap مرتين لإنشاء OnClick له ، وإستبدل الـ procedure الناتج بالبرمجة الموجودة فى القائمة قم بتشغيل البرنامج بضغط F9 ثم إضغط الزر لطباعة الـ bitmap لا تهتم كثيراً إذا كان حجم المخرجات صغيراً هذا يحدث لأنه ، إذا لم تكن قد قست الصور المطبوعة طبقاً لـ resolution الطابعة ، فإن النتائج تستخدم الـ resolution الإفتراضى للطابعة . هذا يمثل مشكلة كبيرة خاصة مع طابعات الليزر ، التى توفر resolution كبيراً جداً عن شاشات العرض الحالية . فعند ٣٠٠ نقطة لكل بوصة ، قد تكون مخرجات الجرافيك صغيرة جداً بحيث لا تقرأ . لقياس الصور بحيث تتماشى بشكل أفضل مع الـ resolution المخرجات ، إستدع الـ Windows API GetDeviceCaps وأطلب قيم الـ logPixelsX والـ logPixelsY لـ device printer context handle . هذا الـ pixel لكل بوصة للجهاز ، والتى ، عند تقسيمها على خاصية الـ form PixelsPerInch ، تقاس الصورة بحجم مناسب .

إن الرقم الحقيقى للـ pixels لكل بوصة فى جهاز معين قد تختلف عن القيمة المنطقية لعدة أسباب ، واحداً منها أن الـ Windows يعرض نص الشاشة أكبر منه فى واقع الأمر ، لذا تكون أحجام النقاط الصغيرة مقروءة على شاشات منخفضة الـ resolution .



على القرص المدمج : تقوم القائمة ١٤-٥ بقياس الصورة المطبوعة طبقاً للطابعه . إستبدل OnClick للبرنامج الإختبارى بهذة القائمة ، أعد التشغيل والطبع . يجب أن يكون حجم الصورة الآن أقرب لمظهرها على

الشاشة . مرة أخرى، هذا لا يعنى أن الشاشة والأحجام المطبوعة تتطابق بالضبط؛ فقط إنها تبدو متشابهة من مسافات الرؤية العادية . هذا النص للقائمة ١٤-٦ موجود على القرص المدمج فى دليل Source\PrintMisc ، فى ملف Print2.pas .

القائمة (١٤-٦): إستخدم هذا الـ code لقياس صورة طبقاً لـ resolution الطابعة.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  ScaleX, ScaleY: Integer;
     R: TRect;
begin
     Printer.BeginDoc;
     with Printer do
     try
       ScaleX :=
          GetDeviceCaps(Handle, logPixelsX) div PixelsPerInch;
            ScaleY :=
          GetDeviceCaps(Handle, logPixelsY) div PixelsPerInch;
            R := Rect(0, 0, Image1.Picture.Width *
       ScaleX,
          Image1.Picture.Height * ScaleY);
            Canvas.StretchDraw(R,
       Image1.Picture.Graphic);
        finally
       EndDoc;
        end;
end;
```

إن الـ procedure الموجود فى القائمة ١٤-٦ يبدأ بإثنين من متغيرات العدد الصحيح، وهى ScaleX و ScaleY ، لقيم تساوى الـ pixels لكل بوصة خاصة بالطابعة التى تم الحصول عليها بواسطة الـ GetDeviceCaps مقسمة على خاصية الـ form PixelsPerInch . هناك خاصية واحدة لأن نظام عرض الـ VGA مربعة، وعدد الـ pixeles لكل بوصة يجب أن يكون واحداً رأسياً وأفقياً. ولا يعتبر نفس الشيء صحيحاً بالضرورة لكل أجهزة المخرجات، وهذا هو السبب فى أن الـ GetDeviceCaps يوفر قيم لمحاور الـ x- والـ y- .

بعد تحديد الـ ScaleX والـ ScaleY، يقوم الـ procedure بإنشاء TRect، R، لأبعاد المخرجات. وبضرب الـ Picture.Width والـ Picture.Height التابعة للـ Image فى عوامل القياس يتم تعديل المستطيل ليتناسب بشكل مقبول. أخيراً، الـ StretchDraw ترسم الـ bitmap بالإشارة إلى خاصية الـ Graphic للـ Picture.

ملحوظة: تبدأ عملية الطباعه فعلياً عندما يستدعى البرنامج الـ EndDoc أو الـ NewPage. إن الرسم بخاصية الـ Printer Canvas لا ينتج أى مخرجات.

طباعة الـ bitmap، والإيقونات، والـ metafile:



على القرص المدمج: لا يجب عليك إستخدام الـ Image لطباعة الجرافيك. يمكنك إستخدام التقنية الموضحة فى القائمة ١٤-٦ لطباعة الـ bitmap والأيقونات والـ meta file. قم بإنشاء TPicture (أنظر الباب السابق لمعرفة التعليمات) وقم بتمرير خاصية الـ Graphic التابعة له للـ Draw methods أو الـ StretchDraw بالـ PrinterCanvas. توضح القائمة ١٤-٧ كيفية إستخدام هذه التقنية لطباعة ملف الـ Sample.bmp على القرص المدمج فى دليل الـ Source\Data. قم بتعديل string اسم المسار كما هى الحاجة فى الإستدعاء للـ Load Fram File. يوجد نص هذا الـ procedure على القرص المدمج فى دليل Source\PrintMisc فى ملف Print2.pas إن القائمتين ١٤-٦ و١٤-٧ تنتجان نفس النتائج. وأى تقنية تستخدم تعتمد على ما إذا كنت تريد إستخدام Image أو إنشاء TPicture تحت تحكم البرنامج. إن عرض الـ Image هو الأسهل غالباً، فإذا كنت لا تستطيع أن تحدد، إستخدم الـ method الموجود فى القائمة ١٤-٦. ولكن، القائمة ١٤-٧ أكثر فاعلية فى إستخدامها للذاكرة لأن bitmap object موجود فقط داخل نطاق الـ event nandler.

القائمة ١٤-٧ إستخدم هذا الـ code لطباعة أى bitmap أو أيقونة أو metafile دون إستخدام Image.

```
procedure TForm1.Button1Click(Sender: TObject);
var
```



```
    P: TPicture;
     ScaleX, ScaleY: Integer;
     R: TRect;
  begin
     P := TPicture.Create;
     try
    // Modify the pathname string in the following statement
     // to load any .bmp, .ico, or metafile:
      P.LoadFromFile('D:\Delphi 4
      Bible\Source\Data\Sample.bmp');
        Printer.BeginDoc;
        with Printer do
        try
         ScaleX :=
            GetDeviceCaps(Handle, logPixelsX) div
          PixelsPerInch;
             ScaleY :=
            GetDeviceCaps(Handle, logPixelsY) div
          PixelsPerInch;
             R := Rect(0, 0, P.Width * ScaleX, P.Height *
        ScaleY);
           Canvas.StretchDraw(R, P.Graphic);
          finally
         Printer.EndDoc;
          end;
     finally
      P.Free;
     end;
  end;
```

طباعة أشكال الجرافيك :

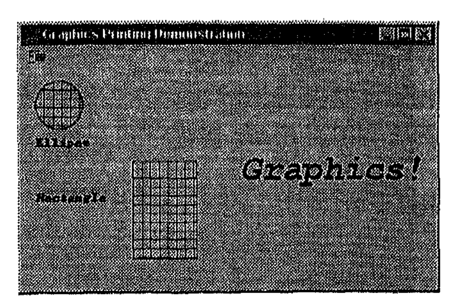


على القرص المدمج : لعرض وطباعة object الجرافيك، أكتب procedure يرسم الجرافيك والنص على Canvas يتم تمريره على أنه argument . إستدع procedure للطباعة على الـ Canvas الخاص بالـ Printer .

إستدع نفس الـ procedure فى OnPaint الخاص بالـ form لعرض الجرافيك. والحيلة هى أن تقيس الصورة المطبوعة فتخرج على حجم معقول للورقة



وعلى الشـاشـة . على القـرص المدمج ، يوضح تطبيـق الـ PrintGr فى دليل الـ Source\PrintGr كيفية طباعة وعرض object الجرافيك والنص يوضح شكل (١٤-٣) عرض الـ PrintGr . توضح القائمة (١٤-٨) source code البرنامج . قم بتشغيل البرنامج ، إختر أمر الـ File\Print للطباعة .



شكل (١٤-٣): يوضح الـ PrintGr إحدى طرق عرض وطباعة objects الجرافيك والنص.

القائمة (١٤-٨): PrintGr\Main.pas.

```
unit Main;

interface

uses
  SysUtils, Windows, Messages, Classes, Graphics,
    Controls, Forms, Dialogs, Menus, Printers;

   type
    TMainForm = class(TForm)
        MainMenu1: TMainMenu;
        File1: TMenuItem;
        Print1: TMenuItem;
        N1: TMenuItem;
        Exit1: TMenuItem;
        PrintDialog1: TPrintDialog;
        procedure Print1Click(Sender: TObject);
        procedure FormPaint(Sender: TObject);
```

```
      procedure Exit1Click(Sender: TObject);
       private
      procedure PaintGraphics(C: TCanvas;
          ScaleX, ScaleY: Integer);
          public
      { Public declarations }
       end;

var
    MainForm: TMainForm;

implementation

{$R *.DFM}

procedure TMainForm.PaintGraphics(C: TCanvas;
 ScaleX, ScaleY: Integer);
var
    R: TRect;
    P: TPoint;

    function ScalePoint(X, Y: Integer): TPoint;
    begin
        Result := Point(X * ScaleX, Y * ScaleY);
    end;

    function ScaleRect(L, T, R, B: Integer): TRect;
    begin
     Result :=
         Rect(L * ScaleX, T * ScaleY, R * ScaleX, B *
        ScaleY);
         end;

begin
 with C do
    begin
```



```
      Pen.Color := clBlue;
        Brush.Color := clRed;
        Brush.Style := bsCross;
        Font.Name := 'Courier New';
        Font.Size := 8;
        Font.Style := [fsBold, fsItalic];
        R := ScaleRect(12, 12, 57, 57);
        Ellipse(R.Left, R.Top, R.Right, R.Bottom);
        R := ScaleRect(100, 85, 160, 174);
        Rectangle(R.Left, R.Top, R.Right, R.Bottom);
        P := ScalePoint(12, 60);
        TextOut(P.X, P.Y, 'Ellipse');
        P := ScalePoint(12, 110);
        TextOut(P.X, P.Y, 'Rectangle');
        Font.Size := 24;
        Font.Style := [fsBold, fsItalic];
        P := ScalePoint(200, 75);
        TextOut(P.X, P.Y, 'Graphics!');
      end;
end;
procedure TMainForm.Print1Click(Sender: TObject);
var
 ScaleX, ScaleY: Integer;
begin
 if PrintDialog1.Execute then
    Printer.BeginDoc;
    try
      ScaleX := GetDeviceCaps(Printer.Canvas.Handle,
           logPixelsX) div PixelsPerInch;
             ScaleY := GetDeviceCaps
      (Printer.Canvas.Handle,
           logPixelsY) div PixelsPerInch;
             PaintGraphics(Printer.Canvas, ScaleX,
      ScaleY);
       finally
      Printer.EndDoc;
```

```
    end;
  end;

  procedure TMainForm.FormPaint(Sender: TObject);
  begin
    PaintGraphics(Canvas, 1, 1);
  end;

  procedure TMainForm.Exit1Click(Sender: TObject);
  begin
    Close;
  end;

  end.
```

يستدعى البرنامج الـ PaintGraphics procedure لعرض object ونصه. يعرف الـ procedure ثلاثة parameters : TCanvas، وأثنين من الـ Integers وهما ؛ الـ ScaleX والـ ScaleY، لقياس الإحداثيات.

وتقوم الـ functions بأداء القياس. تقوم الـ ScalePoint بضرب الـ x والـ y فى عوامل القياس وترجع سجل الـ TPoint بنفس الشئ، ولكن ترجع سجل TRect. ويوضح الـ PaintGraphics كيفية إستخدام هذه الـ functions. على سبيل المثال، لتلوين زر بيضاوى ينفذ البرنامج هذه العبارات :

```
R := ScaleRect(12, 12, 57, 57);
Ellipse(R.Left, R.Top, R.Right, R.Bottom);
```

إن إستدعاء الـ ScaleRect يضبط الإحداثيات لتشمل الـ resolution الأكبر لمعظم الطابعات. ولكن هذه ليست الطريقة الوحيدة لأداء القياس - يوضح الفصل التالى تقنية أخرى.

إنشاء أمر preview للطبع

إذا لم تتطابق resolutions الطابعة والشاشة بالضبط، فمن المستحيل إنتاج الـ object الجرافيكية الموجودة على الشاشة، ولكن بواسطة برمجة حذرة وfont TrueType قابلة للقياس، تكون النتائج جيدة لكثير من التطبيقات.

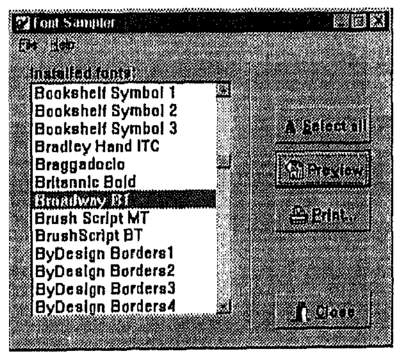




على القــرص المدمج: إن البــرنامج الأخــيــر فى هـذا البــاب يعــرض مـخـرجات الـ WYSIWYG ، ويوضح مزيداً من تقنيات الطبع ، مثل multipage printing وكـيـفـيـة تنفـيـذ نافـذة preview الطبـاعـة .

والبرنامج الذى يسمى FontSamp ، يعرض قائمة من الـ fonts لتجربة البرنامج ، قم بتحميل ملف مشروع الـ FontSamp.dpr من على القرص المدمج فى دليل الـ Source\FontSamp . إضغط F9 للتشغيل وإختر واحداً أو أكثر من أسماء الـ fonts ويمكنك أيضاً ضغط الـ Select all button لتختار كل الـ font ثم تضغط Print لتطبع الصفحات لكل font مختار بأحجام وأساليب متنوعة . إضغط Preview لترى الصفحات قبل الطباعة . يوضح شكل ١٤-٤ العرض الرئيسى للبرنامج . إختر واحداً أو أكثر من أسماء الـ fonts من مربع القائمة .

إضغط Print لطباعة عينات كل font . إضغط Preview لرؤية الصفحات قبل الطبع . يوضح شكل ١٤-٥ نافذة الـ Preview .

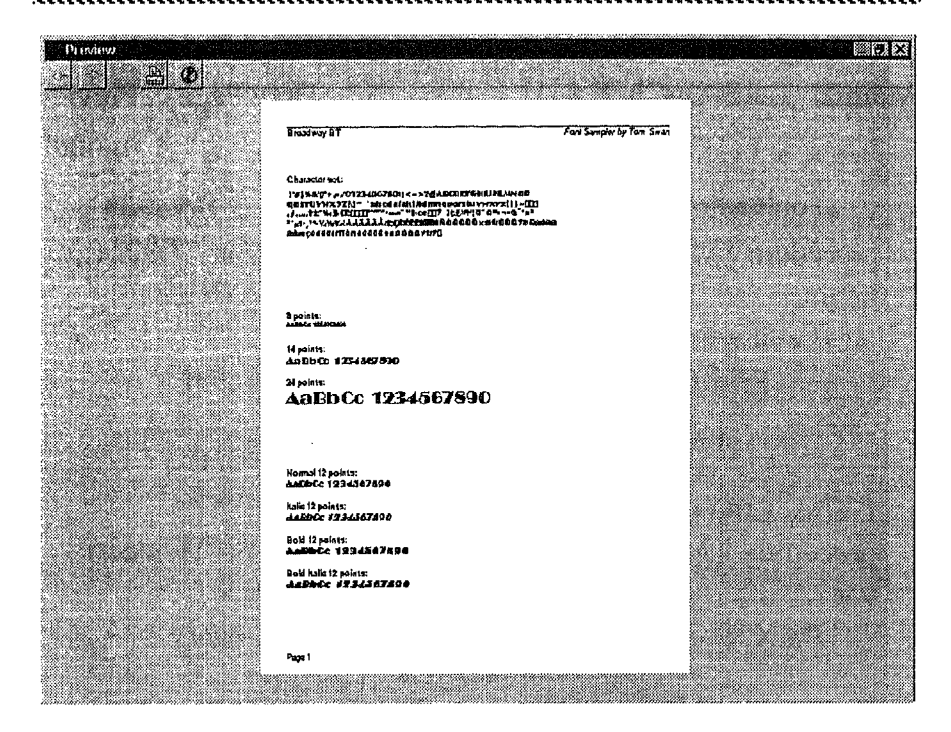


شكل رقم (١٤-٤): العرض الرئيسى للـ FontSamp.



ملحـوظة: تعـرض نـافـذة الـ preview مـراجـعـة الشكل النهـائى لـلـ FontSamp TrueType fonts مثل الـ Arial والذى يتميز بدقة أكثر من الـ bitmap fonts مثل الـ Courier . زد حجم نافـذة الـ preview للحصول على أفضل النتائج .

تقوم الـ TBitmap class الخاصة بـ Delphi بتبسيط الخطوات اللازمة لإنشاء نافذة preview . يقوم الـ FontSamp بإنشاء TBitmap . وعرض وطول محدد بنفس النسبة مثل الـ ٨١/٢ × ١١ بوصة للصفحة .



شكل ١٤-٥ نافذة الـ FontSamp preview توضح الـ Broadway BT font من نوع الـ TrueType.

لمعاينة المخرجات المطبوعة، يرسم البرنامج على الـ Canvas الخاص بالـ bitmap ولطباعتهما، يرسم البرنامج على الـ Canvas الخاص بالـ Printer. وسوف تكون النتيجة تمثيل بصرى حقيقى للمخرجات المطبوعة للبرنامج.

وغالباً من الناحية العملية، من الصعب تطابق المخرجات المطبوعة بدقة فى bitmap صغيرة. وتعتبر الـ Fonts غير الـ TrueType مزعجة ولا يتم إعادة تحديد حجمها بشكل جيد. وإحدى الخطط التى تبدو ناجحة، والتى لا تتطلب أشهر طويلة، هى برمجة كل الإحداثيات كقيم floating-point لتمثل البوصات. هذه الخطة تحافظ على برنامج منظم وبسيط. على سبيل المثال، لطباعة نص فى منتصف صفحة، بفرض أن البعد الرأسى يساوى ١١ بوصة، فإنك تختار إحداثية الـ y مساوية لـ ٥,٦ بوصة. كل ما تحتاجه هو بعض الـ functions البسيطة لتحويل البوصات إلى pixels مؤسسة على الـ resolution المنطقى للـ Canvas.



وهناك ثلاثة تراكـيب فى FontSamp تنفــذ هذه الخطة وهى : Main ، Preview ، وDrawPage . توضح القـائمة ١٤-٩ OnClick التابعين للـ Main .module

القائمة ١٤-٩، لزرى الـ Print والـ Preview للـ FontSamp\Main.pas..

```
procedure TMainForm.PreviewBitBtnClick(Sender: TObject);
begin
 if PreviewForm.ShowModal = mrOk then
      PrintBitBtn.Click;  { Preview's print button }
end;

{ This is the procedure that prints the pages }
procedure TMainForm.PrintBitBtnClick(Sender: TObject);
var
   PpiX, PpiY, Page, FirstPage, LastPage: Integer;

   { Initialize PrintDialog1 object}
   procedure InitPrintDialog;
   begin
    with PrintDialog1 do
      begin
       MinPage := 1;
         MaxPage := FontListBox.SelCount;
         FromPage := MinPage;
         ToPage := MaxPage;
        end;
   end;

   { Initialize printing variables }
 procedure InitParameters;
 begin
 { Do our own scaling based on Page width and height.
This
```



```
  seems to be more reliable than GetDeviceCaps. }
    PpiX := Trunc(Printer.PageWidth / 8.5);
      PpiY := Trunc(Printer.PageHeight / 11.0);
     { Set FirstPage and LastPage }
    if PrintDialog1.PrintRange = prAllPages then
    begin
       FirstPage := 1;
         LastPage := FontListBox.SelCount;
        end else
      begin
       FirstPage := PrintDialog1.FromPage;
         LastPage := PrintDialog1.ToPage;
        end;
     end;

  begin
   InitPrintDialog;
     if PrintDialog1.Execute then
     begin
      Printer.BeginDoc;
        try
         InitParameters;
           for Page := FirstPage to LastPage do
           begin
             DrawOnePage(Printer.Canvas, FontListBox,
          Page,
               Printer.PageWidth, Printer.PageHeight,
             False,
                 PpiX, PpiY);
                 if page < LastPage then
               Printer.NewPage;
                 end;
         finally
        Printer.EndDoc;
```



```
        end;
          end;

  end;
```

الـ Preview OnClick يعـرض الـ ShowModal . إذا رجـعـت هذه الـ function بـ mrOK، يكون المستخدم قد ضغط زر الـ Preview Print من أجل الطباعة . يستدعى Print Click method. procedure .

يقوم الـ Print OnClick بإستدعاء procedures منطقيين لبدء dialog الطباعة و parameters أخرى . حدد خصائص الـ MinPage والـ MaxPage للـ PrintDialog1 بمدى الصفحة . فى هذه الحالة ، هناك font واحد لكل صفحة ، لذا يكون أكبر عدد من الصفحات مساوياً للـ FontListBox.SelCount حدد الـ FromPage والـ ToPage بنفس هذه القيم حتى تظهرا فى نوافذ الـ dialog الطبع .

لبدء الطباعة ، يحدد الـ InitParameters procedure متغيرات الـ PpiX والـ PpiY والتى تكون مساويه مساوية لعدد الـ pixels المنطقية لكل بوصة لحجم الصفحة المعنية ذات الـ ٥ ,٨ × ١١ بوصة .

إن عمل الحسابات بهذه الطريقة يبدو أكثر دقة من إستدعاء الـ GetDeviceCaps . ويقوم الـ procedure يفحص خاصية الـ PrintRange للـ PrintDialog1 . فإذا كانت مساوية للثابت prAllPages ، يكون المستخدم قد ضغط الـ All radio button الخاص بالـ dialog ؛ وإلا ، يكون المستخدم قد قام بتعديل مدى الصفحة ويمكنك الحصول على هذه القيم من الـ ToPage والـ FromPage كما تم التوضيح .

وبعد كل هذا ، يعتبر البرنامج مستعداً للطباعة . وتتولى for loop هذا الأمر بإستدعاء procedure ، وهو الـ DrawOnePage (فى DrawPage module) مع arguments متنوعة . وأكثر أهمية هى الأولى ، Printer.Canvas ، والتى توجه المخرجات إلى الطابعة . أما الـ Printer.PageWidth والـ Printer.PageHeight فيعطيان الـ DrawOnePage حدود المخرجات لصفحة واحدة . إن إستدعاء الـ Printer.NewPage ينقلك إلى صفحة جديدة بعد الرسم .



إن الـ Preview module في القـــائمــــة (١٤-١٠) تســـتـــدعى الـ DrawOnePage لتحاكي الصفحات المطبوعة . وهذا الـ module قصيرة نسبياً، لذا ذكرتها كاملة هنا .

القائمة ١٤-١٠: FontSamp\Preview.pas

```
unit Preview;

interface

uses
  SysUtils, Windows, Messages, Classes, Graphics,
    Controls, Forms, Dialogs, StdCtrls, Buttons,
  ExtCtrls,
    DrawPage;

type
  TPreviewForm = class(TForm)
    ToolBar: TPanel;
      LeftPageSB: TSpeedButton;
      RightPageSB: TSpeedButton;
      PrintSB: TSpeedButton;
      CloseSB: TSpeedButton;
      procedure FormCreate(Sender: TObject);
      procedure FormResize(Sender: TObject);
      procedure FormPaint(Sender: TObject);
      procedure FormDestroy(Sender: TObject);
      procedure CloseSBClick(Sender: TObject);
      procedure PrintSBClick(Sender: TObject);
      procedure FormActivate(Sender: TObject);
      procedure LeftPageSBClick(Sender: TObject);
      procedure RightPageSBClick(Sender: TObject);
     private
    PreBits: TBitmap;      { Preview bitmap with
    canvas }
      PpiX, PpiY: Integer;    { Logical pixels per inch
    }
```



```
      Page: Integer;            { One font sampler per page }
        procedure InitGlobals; { Initialize global
      variables }
       public
        FontListBox: TListBox; { Reference to form's
      ListBox }
     end;

var
 PreviewForm: TPreviewForm;

implementation

uses Main;

{$R *.DFM}

const
 border = 10;    { Top and bottom preview bitmap borders }

{ Create form and a bitmap to represent the preview page }
procedure TPreviewForm.FormCreate(Sender: TObject);
begin
 FontListBox := nil;
   PreBits := TBitmap.Create;
end;

{ Initialize global variables and window size }
procedure TPreviewForm.InitGlobals;
begin
 PreBits.Width :=
    ClientWidth div 2; {Bitmap width = @bf1/2 client
    width }
     PreBits.Height :=
    Round(1.3 * PreBits.Width);  { 1.3 = 8@bf1/2 x 11
    ratio }
     PpiX :=
```



```
      Round(PreBits.Width / 8.5);    { Logical pixels per inch }
       PpiY :=
      Round(PreBits.Height / 11.0);  { Logical pixels per
      inch }
       if WindowState <> wsMaximized then   { Adjust window
   bottom }
      ClientHeight :=
         ToolBar.Height + PreBits.Height + border * 2;
end;

{ Tip: OnResize is called before OnActivate, but
only if the form is NOT maximized, in which case FormResize
is never called. Don't use OnResize as your only
display initializer-also initialize in OnActivate. }
procedure TPreviewForm.FormResize(Sender: TObject);
begin
   InitGlobals;
   DrawOnePage(PreBits.Canvas, FontListBox, Page, {Redraw
  page}
     PreBits.Width, Height, True, PpiX, PpiY);
       Invalidate;
end;

{ Because the program does its own scaling, we can call Draw
instead of StretchDraw as some previewers do. This keeps the
display fast and keeps the text looking as WYSIWYG as possible}

procedure TPreviewForm.FormPaint(Sender: TObject);
begin
   Canvas.Draw(ClientWidth div 4,
     ToolBar.Height + border, PreBits);
end;

procedure TPreviewForm.FormDestroy(Sender: TObject);
begin
   PreBits.Free;
```



```
end;

procedure TPreviewForm.CloseSBClick(Sender: TObject);
begin
  ModalResult := mrCancel;
end;

procedure TPreviewForm.PrintSBClick(Sender: TObject);
begin
  ModalResult := mrOk;
end;

{ This procedure prepares the FontListBox, and it draws
the first page (or a blank if no font is selected). The
procedure also enables and disables the toolbar SpeedButtons }
procedure TPreviewForm.FormActivate(Sender: TObject);
begin
{ If you don't assign a ListBox to PreviewForm.FontListBox,
this statement picks up the ListBox from the parent form. }
      if FontListBox = nil then
        FontListBox := MainForm.FontListBox;
{ Draw first page }
  Page := 1;
      InitGlobals;
      DrawOnePage(PreBits.Canvas, FontListBox, Page,
        PreBits.Width, Height, True, PpiX, PpiY);
{ Enable / disable speed buttons in toolbar }
with FontListBox do
  begin
    LeftPageSB.Enabled := SelCount > 1;
        RightPageSB.Enabled := SelCount > 1;
        PrintSB.Enabled := SelCount > 0;
      end;
end;

{ Display previous page }
```

```
procedure TPreviewForm.LeftPageSBClick(Sender: TObject);
begin
 if Page > 1 then
    begin
     Dec(Page);
        DrawOnePage(PreBits.Canvas, FontListBox,
     Page,
         PreBits.Width, Height, True, PpiX, PpiY);
           Invalidate;
    end;
end;

{ Display next page }
procedure TPreviewForm.RightPageSBClick(Sender: TObject);
begin
 if Page < FontListBox.SelCount then
    begin
     Inc(Page);
        DrawOnePage(PreBits.Canvas, FontListBox,
     Page,
         PreBits.Width, Height, True, PpiX, PpiY);
           Invalidate;
    end;
end;

end.
```

تعـرف الـ Preview module مـتـغـيـرات مـتـعـددة خـاصـة بالـ TPreviewForm class . ويمثل الـ PreBits، وهو أحد الـ TBitmap object، صفحة واحدة من الـ preview . وتعتبر الـ PpiX والـ PpiY هي الـ pixels المنطقية لكل بوصـة في هذه الـ bitmap . أمـا الـ Page فـهي رقم الصـفـحـة والـ FontListBox هو مرجع للـ form الأم FontListBox . ويقوم الـ Procedure InitGlobals ببدء هذه المتغيرات .

وتقـوم الـ Preview form بإنشـاء PreBits bitmap في الـ OnCreate ويوضح الـ InitGlobals procedure كـيـفـيـة أداء بعض الحـسـابات الضـرورية

لصفحة المخرجات المقلدة إن تعيين قيم للـ PreBits.Height والـ PreBits.Width يحدد حجم bitmap بالنسبة لنافذة الـ Preview، ولكن بالنسبة لـ ٢/ ٨١ × ١١ بوصة كحجم الصفحة. يحدد البرنامج الـ PpiX والـ PpiY بالـ pixels لكل بوصة لنفس حجم هذه الصفحة - وهذا لا يساوى الـ pixels لكل بوصة على الشاشة أو على الـ form ولكن يساوى الـ bitmap التى تحاكى صفحة المخرجات. أخيراً، إذا لم يتم تكبير النافذة، يعين البرنامج قيمة معدلة لخاصية الـ ClientHeight للـ form. هذا ينقل زر النافذة حتى تصبح صفحة الـ Preview bitmap مرئية تماماً دائماً.

حاول أن تعيد تحديد حجم نافذة الـ Preview وأنظر كيف يحاول زر النافذة أن يبحث عن مستواه.

إنتقل إلى الـ FormPaint OnPaint. لعرض صفحة الـ Preview يمرر البرنامج PreBits إلى الـ Canvas Draw method. هذا يجعل المخرجات سريعة. ويستطيع أمر Preview للطبع أن يشكل bitmap بالحجم الحقيقى ثم يستخدم الـ StretchDraw بعد ذلك لعرض هذه الـ bitmap فى نافذة. وهذا أسهل فى البرمجة، ولكنه يعطى نتائج ضعيفة.

ولأن البرنامج يستخدم الـ Draw، فيجب عليه أن يعيد رسم كل صفحة عندما:

* تظهر نافذة الـ Preview لأول مرة.

* يتغير حجم النافذة.

* يطلب المستخدم صفحة مختلفة.

إن الـ OnResize، FormResize، يحدد متغيرات عمومية طبقاً لحجم النافذة الحالى، ويستدعى الـ DrawOnePage لتشكيل مخرجات الـ preview، للرسم على الـ bitmap البعيدة عن الشاشة، يمرر البرنامج الـ PreBits.Canvas للـ DrawOnePage ثم يستدعى بعد ذلك الـ Invalidate، مما يجعل الـ Windows يصدر رسالة الـ wm_Paint للنافذة هذا يؤدى للـ OnPaint، والذى، كما رأيت، يرسم الـ bitmap بإستدعاء الـ Canvas.Draw method (تلك الموجودة فى الـ Canvas التابع للـ form وليس التابع لـ bitmap).



عندما يتم تنشيط الـ Preview form لأول مرة، ينشئ الـ OnActivate بعض القيم مثل الـ FontListBox reference ثم تستدعى بعد ذلك الـ DrawOnePage لتكون صفحة المخرجات الأولى. ليس من الضرورى أن تستدعى الـ Invalidate event لإيجاد الـ OnPaint، لأن هذا يحدث بصورة تلقائية بعد الـ OnActivate. إن OnActivate يقوم أيضاً بإبطال وتشغيل الـ SpeedButtons الخاصة بنافذة الـ Preview إعتماداً على عدد الـ fonts المختارة (الـ FontListBox.SelCount).

يقوم الـ FormActivate procedure بتعيين الـ FontListBox للـ MainForm FontListBox وهذا التعيين لا يؤدى إلى إنشاء list box آخر. إستخدم nil، كما هو موضح هنا كـ flag، للإشارة إذا ما كان مرجعاً مثل الـ FontListBox قد تم بدءه. على سبيل المثال، يمكنك إدخال أثنين أو أكثر من الـ list box مع مجموعات fonts مختلفة فيهم وعين الـ reference لهم بـ PreviewForm.FontListBox.

أو يمكنك أن تجعل الـ module تفحص هل الـ FontListBox محدد بـ nil، فى هذه الحالة تنتقى الـ FontListBox الخاص بالـ form الأم طبقاً للنظام الإفتراضى للحاسب. (يتم إنشاء الـ FontListBox بـ nil فى الـ OnCreate الخاص بالـ form). وهذا المثال جيد على البرمجة الدفاعية - إن الـ module تعمل بشكل صحيح بغض النظر عما إذا كانت هناك module أخرى تنشئ متغير الـ FontListBox العام.

أخيراً، إن الـ SpeedButtons OnClick الخاصة بالصفحة التالية والسابقة للـ tools bac توجد فى الـ Preview module. إن الـ procedures تقوم بزيادة أو تقليل متغير الـ Page وتستدعى الـ DrawOnePage لصنع الـ bitmap.

كما أوضحت، إن إستدعاء الـ Invalidate يصر على حدوث الـ OnPaint event، الذى يرسم الـ bitmap الجديدة على الـ Canvas الخاص بالـ Preview form.

كما أدركت حتى الآن، أن الـ DrawOnePage يربط المخرجات الجرافيكية لهذا البرنامج مع بعضها البعض. وتوضح القائمة ١٤-١١ الـ module DrawPage التى تنفذ هذا الـ procedure.

القائمة ١٤-١١: FontSamp\DrawPage.pas

```
unit Drawpage;

interface

uses SysUtils, Graphics, StdCtrls;

{ Call DrawOnePage to form each sampler page either
during
printing or for previewing with an offscreen bitmap. }

procedure DrawOnePage(
 Canvas: TCanvas;          { Printer or TBitmap Canvas for preview
 }
  FontListBox: TListBox; { Fonts with multiple
 selections }
  Page,            { Page number (FontList selection
 index) }
  PageWidth,              { Unscaled page width in
 pixels }
  PageHeight: Integer;    { Unscaled page height in
 pixels }
  Previewing: Boolean;   { True if previewing; else
 printing }
  PpiX, PpiY: Integer      { Pixels per inch on X- and Y-
 axes }
 );

implementation

uses Main;

var
  C: TCanvas;
  FontName, HeaderName: String;
  PixelsPerInchX, PixelsPerInchY: Integer;
  Preview: Boolean;
```

```
{ Return selected font at index }
function SelectedFont(ListBox: TListBox;
 Index: Integer): String;
var
 I: Integer;
begin
    with ListBox do
    for I := 0 to Items.Count - 1 do
    if Selected[I] then
    begin
     Dec(Index);
        if Index <= 0 then
        begin
         Result := Items[I];
            Exit;
           end;
    end;
    Result := 'System';
end;

{ Assign font name, style, and size to Canvas font }
procedure SetFont(const Name: String; Style: TFontStyles;
 Size: Integer);
begin
{ Adjust point size for preview page's logical pixels per
inch relative to the form's actual pixels per inch. This allows
the program to draw into the bitmap with TextOut, and then
display the bitmap in real size with Canvas.Draw. Some print
previewers use StretchDraw, which produces relatively poor
results.}
 if Preview then
      Size := Round(Size *
         (PixelsPerInchY / MainForm.PixelsPerInch));
{ Assign parameters to Canvas C Font property }
 C.Font.Name := Name;
    C.Font.Style := Style;
```



```
    C.Font.Size := Size;
end;

{ Return pixel width of Name in inches }
function InchWidth(const Name: String): Double;
begin
  Result := C.TextWidth(Name);
    Result := Result / PixelsPerInchX;
end;

{ Return pixel height of Name in inches }
function InchHeight(const Name: String): Double;
begin
  Result := C.TextHeight(Name);
    Result := Result / PixelsPerInchY;
end;

{ Write string S at inch coordinates X and Y }
procedure TextAtInch(X, Y: Double; const S: String);
var
  Px, Py: Integer;
begin
  Px := Round(X * PixelsPerInchX);
    Py := Round(Y * PixelsPerInchY);
    C.TextOut(Px, Py, S);
end;

{ Draw a line at inch coordinates X1, Y1, X2, Y2 }
procedure LineAtInch(X1, Y1, X2, Y2: Double);
var
  Px1, Py1, Px2, Py2: Integer;
begin
  Px1 := Round(X1 * PixelsPerInchX);
    Py1 := Round(Y1 * PixelsPerInchY);
    Px2 := Round(X2 * PixelsPerInchX);
    Py2 := Round(Y2 * PixelsPerInchY);
```

```
    C.MoveTo(Px1, Py1);
     C.LineTo(Px2, Py2);
end;

{ Draw header at top of page }
procedure DrawHeader(const Name: String);
var
  S: String[24];
begin
     SetFont(HeaderName, [fsBold], 12);
     TextAtInch(0.5, 0.5, Name);
     SetFont(HeaderName, [fsItalic], 12);
     S := 'Font Sampler by Tom Swan';
     TextAtInch(8.0 - InchWidth(S), 0.5, S);
     LineAtInch(0.5, 0.5, 8.0, 0.5);
end;

{ Draw footer at bottom of page }
procedure DrawFooter(Page: Integer);
begin
  SetFont(HeaderName, [], 12);
     TextAtInch(0.5, 10.5, 'Page ' + IntToStr(Page));
end;

{ Draw sample character set (ASCII 32-255) }
procedure DrawCharacterSet;
var
     H: Double;
     procedure DrawOneLine(J, K: Integer);
     var
       I: Integer;
          S: String;
        begin
       S := '';
          for I := J to K do
            S := S + Chr(I);
```



```
        TextAtInch(0.5, H, S);
          H := H + InchHeight('M');
        end;
begin
    SetFont(HeaderName, [fsBold], 12);
    TextAtInch(0.5, 1.4, 'Character set:');
    SetFont(FontName, [], 10);
    H := 1.5 + InchHeight('M');
    DrawOneLine(32, 80);
    DrawOneLine(81, 129);
    DrawOneLine(130, 178);
    DrawOneLine(179, 227);
    DrawOneLIne(228, 256);
end;

{ Draw sample text in 8, 14, and 24 point sizes }
procedure DrawPointSamples;
var
 H, M: Double;
    procedure DrawOneSample(Pts: Integer);
    begin
      SetFont(HeaderName, [fsBold], 12);
        TextAtInch(0.5, H, IntToStr(Pts) + ' points:');
        M := InchHeight('M');
        H := H + M;
        SetFont(FontName, [], Pts);
        TextAtInch(0.5, H, 'AaBbCc 1234567890');
        H := H + M * 2;
      end;
begin
    H := 4.0;
    DrawOneSample(8);
    DrawOneSample(14);
    DrawOneSample(24);
end;
```

```
{ Draw normal, italic, bold, and bold-italic samples }
procedure DrawNormBoldItal;
var
    H, M: Double;
    procedure DrawOneLine(const S: String; Style:
  TFontStyles);
    begin
      SetFont(HeaderName, [fsBold], 12);
        TextAtInch(0.5, H, S);
        M := InchHeight('M');
        H := H + M;
        SetFont(FontName, Style, 12);
        TextAtInch(0.5, H, 'AaBbCc 1234567890');
        H := H + M * 2;
    end;
begin
  H := 7.0;
    DrawOneLine('Normal 12 points:', []);
    DrawOneLine('Italic 12 points:', [fsItalic]);
    DrawOneLine('Bold 12 points:', [fsBold]);
    DrawOneLine('Bold Italic 12 points:', [fsBold,
  fsItalic]);
end;

{ Printing and preview code calls this procedure to draw
each page. See declaration at top of file for descriptions
of the parameters. }
procedure DrawOnePage(Canvas: TCanvas; FontListBox: TListBox;
  Page, PageWidth, PageHeight: Integer; Previewing:
  Boolean;
    PpiX, PpiY: Integer);
begin
{ Save some parameters in global variables for easy access }
    C := Canvas;
    C.Pen.Color := clBlack;
    PixelsPerInchX := PpiX;
```



```
    PixelsPerInchY := PpiY;
     Preview := Previewing;
  { Draw the font samples on the Canvas (Printer or Preview) }
     with Canvas do
     begin
       FillRect(ClipRect);    { Erase page }
          if (FontListBox = nil) or (FontListBox.SelCount <
        1) then
            Exit; { Display / print blank page if no font
           selected }
               FontName := SelectedFont(FontListBox,
    Page);
       HeaderName := 'Arial';
       DrawHeader(FontName);  { These statements draw
    the }
       DrawFooter(Page);      { header, footer, and font
    samples }
       DrawCharacterSet;
       DrawPointSamples;
       DrawNormBoldItal;
     end;
  end;

   end.
```

إن تقرير كيفية تنفيذ procedures الرسم الخاصة ببرنامجك أمر يرجع إليك، ولكنني أخصصها دائماً لـ module منفصلة. بالإضافة إلى أن البرنامج أسهل في الحفظ، فإن فصل عبارات الجرافيك قد ساعدتني على حمل العديد من البرامج لنظم تشغيل متعددة، والتي يكون لها أوامر رسم مختلفة. إن module DrawPage لديها public procedure واحد، وهو الـ DrawOnePage، والذي كما رأيت، يقوم الـ FontSamp بإستدعاءه لطباعة صفحات البرنامج في تطبيقاتك، قم بتصميم modules الرسم وpreview قبل تنفيذ أوامر الطباعة الخاصة بالبرنامج. هذا يوفر الأوراق ويجبرك على أن تفكر جيداً في كيفية قياس المخرجات للـ resolutions المختلفة. ولأنني إتبعت هذه الخطة في كتابة الـ FontSamp، فإن أوامر الطباعة الخاصة به عملت من المرة الأولى

التى جربتها فيها . (حسناً، ربما كانت من المرة الثالثة، ولكن على أية حال، إن أوامر الطباعة قد بدت أسهل فى الإنشاء لأننى قد حللت المشكلات الصعبة فى الـ DrawPage modules والـ Preview) إننى لم أشرح كل الـ module code DrawPage، والتى لا تحتوى أى عبارات طباعة . ولكن، تكمن أهمية الـ mod-ule فى أنها ترسم على الـ Canvas، بغض النظر عن نوعه . لذلك، يستطيع البرنامج تمرير الـ Canvas الخاص بالـ Printer للـ DrawOnePage ليطبعه، أو الـ Canvas الخاص بالـ PreBits bitmap لرسم صفحات offscreen preview .

إن البرمجة ستصبح منظمة إذا كانت العبارة السابقة صادقة بنسبة ١٠٠٪ . ولكن، من الناحية العملية، لا يكون التوصل إلى مخرجات الـ WYSIWYG بهذه السهولة . لقد إحتجت إلى flag، وهو الـ previewing، لأحدد ما إذا كان الـ mod-ule قد طبع أو تم عرضه إن الـ SetFont Procedure يستخدم هذا الـ flag لتقليل حجم نقطة الـ flag عند الـ preview، وهذا أمر ضرورى لأن النص سوف يتم تحديده بشكل طبيعى لقياس النافذة (وليس قياس الـ bitmap) وللتغلب على هذه المشكلة يستطيع البرنامج رسم fonts فى أحجامها الطبيعية ويستدعى الـ StretchDraw bitmap ولكن، كما ذكرت، تكون النتائج ضعيفة ويتم عرضها ببطء .

افكار للمستخدم الخبير

* إذا كان النص لا يطبع بحجمه المتوقع، تأكد من أن الـ code تستدعى الـ Print.Begin Doc قبل تغيير الـ Printer Canvas Font والخصائص الفرعية له .

* يجب عليك فى النهاية أن تستدعى الـ CloseFile لمتغير الـ System.Text الذى تم تمريره للـ AssignPrn . إن محاولة تعيين ملف مخرجات ثانى دون إغلاق الملف المعين حالياً يؤدى إلى exception .

* عند طباعة plain text، إنتظر حتى تتغير خاصية الـ PageNumber التابعة للـ Printer بعد كل إستدعاء للـ Writeln . عندما تتقدم الـ PageNumber، تكون قد بدأت صفحة جديدة . بدلاً من حساب عدد السطور لكل صفحة كما يقترح هذا الباب، يمكنك إستخدام هذه التقنية لكتابة رأس سطر فى أعلى كل صفحة جديدة .

* عين string لخاصيـة الـ Printer Title لتعـريف مـهـمـة الطبـاعـة فى الـ Windows Print Manager وعلى عناوين صفحة الشبكة .

* لإنشـاء abort printing dialog ، إستـخـدم form منفـصلة وإسـتـدع Show method لها ليعرضها . إجعل الـ form تحدد المتغير العام بـ Ture كـ flag يشـيـر إلى أن المسـتـخـدم قـد ضـغـط الـ Abort . إبحث عن هذا الـ flag داخل الـ printer output loop لبـرنامـجك وإذا كـان Ture flag ، إسـتـدع الـ Printer.Abort method لإلغاء الطباعة . لا يجب عليك إستدعاء الـ EndDoc فى هذه الحالة .

* لفحص ما إذا كانت عملية الطبع قد تم إبطالها (على سبيل المثال ، بواسطة الـ print loop التى تسـتـدعى الـ Printer.Abort) ، إفـحـص الـ flag Read only Printer.Aborted.

المشروعات التى يمكنك تجربتها

١٤-١ : قم بتحسين برنامج العرض Lister بإضافة أمر قائمة لإختيار الـ font والنمط والحجم ، ولجعل أرقام السطور إختيارية . يمكنك أيضاً تنفيذ أمر الـ File|Page Setup لتشكيل العناوين ، أرقام الصفحات ، وما إلى ذلك . إطبع هذه البنود كما تفعل مع العناصر الأخرى بإستخدام الـ Write والـ Writeln . (ملحوظة : إستخدم الـ method الموضح بواسطة تطبيق الـ PrnInfo لتحسب عدد السطور فى كل صفحة) .

١٤-٢ : إن تغيير الـ fonts فى الـ PrnInfo يجعل البرنامج يطبع تقريره بنفس هذا الـ font ، مما قد يسبب عدة مشاكل - على سبيل المثال ، إذا إخترت حجم نقطة كبير أو بنط رمزى راجع هذه الميزة بكتابة procedure يعرض إحصائيات الطابعة لأى font ونمط وحجم نقطة . ثم ، إستخدم procedure لطباعة التقرير بالـ Font المعيارى .

١٤-٣ : أكتب utility لتحدد عدد السطور لكل صفحة لكل font مركب . إطبع تقريراً عن هذه المعلومة .

١٤-٤ : قم بتصميم برنامج طابعة تشخيص يطبع نمط إختيارى، يعتبر هذا البرنامج مقيداً فى فحص حدة الحافة ، مظهر النص ، إعتدال الخط ، ومعلومات أخرى مفيدة .

١٤-٥ : متقدم . قم بتصميم وتنفيذ موثق مشروع Delphi الذى يطبع كل ملفات الـ pas . فى الدليل الحالى وكذلك ملف مشروع dpr. . وكذلك إطبع أيقونات وملفات bitmap .

ملخص :

* إستخدم الـ PrintDialog والـ PrinterSetupDialog لحث المستخدمين على الطباعة وتوفير الوصول إلى إعدادات برنامج تشغيل جهاز الطابعة المركبة .

* إن الـ TPrinter ، الذى لا يوجد على لوحة الـ VCL ، يوفر إمكانيات طباعة . أضف Printer لأمر الـ uses الخاص ببرنامجك . يمكنك عندئذ تعيين قيم لخصائص الـ Printer العام وإستدعاء methods مثل الـ Printer.BeginDoc والـ Printer.NewPage . إنك لا تنشئ أبداً object من الـ TPrinter class المتوفر بالـ Printers unit .

* توجد طريقتان للطباعة : لطباعة نص عادى ، إستخدم ملف الـ Text مع الـ Write procedures والـ Writeln الخاصة بـ Pascal . لطباعة الجرافيك ونص الـ WYSIWYG ، إستخدم الـ Printer Canvas . إستدع الـ BeginDoc . لبدء عملية الطباعة . إرسم على الـ Canvas كما تفعل لعرض الجرافيك فى نافذة . إستدع الـ NewPage للإنتقال إلى صفحة جديدة . إستدع الـ EndDoc لإنهاء الطباعة .

* لطباعة form إستدع الـ Print method الخاص بها ، والذى يحدد مساحة الـ form client area على offscreen bitmap . يستخدم الـ method تقنيات الـ BeginDoc والـ EndDoc التابعة للـ Printer لطباعة الناتج .

* قم بإنشاء أمر preview للطبع وذلك بفصل عبارات مخرجات الجرافيك فى الـ module، أو بواسطة procedure مثل الـ DrawOnePage فى تطبيق الـ

FontSamp . للطباعة ، قم بتمرير الـ Printer Canvas إلى procedure . لرسم صفحات preview مقلدة ، قم بتمرير Canvas خاص بالـ TBitmap وارسم bitmap بإستخدام الـ form Canvas.Draw method (وليس الـ bitmap) .

إن الـ Multiple Document Interface ، أو الـ MDI ، كانت واحدة من الميزات التابعة للـ Windows والمختلف عليها فيما بين واضعى البرامج منذ اليوم الأول لها . إن الـ MDI ليست مناسبة لكل البرامج ، ولكنها مفيدة فى إنشاء نوافذ متعددة الصفحات بإستخدام واجهة تطبيق معيارية ، كما هو موضح فى الباب التالى .

الباب الخامس عشر
تطوير تطبيقات الـ MDI

محتويات هذا الباب:

- Components.
- أسس برمجة الـ MDI.
- Child windows.
- تقنيات MDI أخرى.

إن مطورى الـ Windows يستمتعون بعلاقة الحب/ الكراهية للـ MDI، أو الـ Multiple Document Interface. فى الواقع، إن أغلب واضعى البرامج يحبون أن يكرهوا الـ MDI. عندما تحتاج إطار عمل لإدارة وثائق متعددة، مع قالب للبيانات واحد لكل نافذة، فإن الـ MDI توفر واجهة تطبيق معيارية يسهل إستخدامها، وبمساعدة Delphi، الأكثر مباشرة للبرنامج.

إن هذا الباب يشرح كيفية إستخدام Delphi forms لإستخدام لإنشاء نوافذ الـ MDI الرئيسية والـ child. سوف أشرح أيضاً موضوعات مثل كيفية إنشاء قائمة Window بأوامر إدارة النافذة، وكيفية تقسيم الـ forms إلى classes فرعية لإنشاء child window، كيفية إضافة عناوين نافذة لقائمة Window، وكيفية دمج قوائم النافذة الرئيسية والصغيرة.

Components:

فيما يلى بعض الـ Delphi components لتطوير تطبيقات الـ MDI:

* TForm - هذا الـ component وهو نفسه كالذى تم إستخدامه فى تطبيقات الـ Single Document Interface (SDI)، غير موجود على لوحة الـ

VCL . إن تطبيقات الـ MDI تستخدم objects من الـ TForm class للنوافذ الرئيسية والـ child windows. حدد خاصية الـ FormStyle لنافذة رئيسية بـ fsMDIForm . للـ Child windows، حدد هذه الخاصية بـ fsMDIChild وإتبع التعليمات الموجودة فى هذا الباب لإنشاء Child window فى وقت التشغيل - إستجابة لأمر الـ File|Open، مثلاً. Palette: لا يوجد.

* MainMenu - إن نافذة الـ form الرئيسية لكل تطبيق MDI يجب أن يكون لها MainMenu object، والذى تم تقديمه فى الباب الخامس إن أغلب الـ MDI MainMenu لها عناصر قائمة File و Window، بالرغم من أنك حر فى تسمية قوائمك بما تشاء. ويمكن أيضاً لنوافذ الـ MDI الصغيرة MainMenu objects للدمج فى قائمة النافذة المرئية الـ Standard: Palette.

Note
ملحوظة: إنطق الـ MDI على أنها "M," "D," "I,"، حتى تتفادى الخلط بينها وبين الـ MIDI، وهو واجهة تطبيق لآلة موسيقية. إن الـ MDI والـ MIDI لا علاقة لهما ببعضهما.

أسس برمجة الـ MDI

إن كل تطبيق MDI له ثلاثة أجزاء أساسية : -

* form النافذة الرئيسية للـ MDI .

* وثيقة واحدة أو أكثر للـ child window MDI form .

* القائمة الرئيسية للـ MDI .

على عكس ما يحدث فى برمجة الـ Windows التقليدية، تأخذ Delphi form object محل الـ MDIframe المعيارية والـ Client window. فى العادى، تكون نافذة الإطار هى المرئية؛ والـ client windows تعتبر نوع من الشريك الصامت الذى يتولى العمليات العامة، ينشئ النوافذ الـ child، ويؤدى خدمات الرسالة. فى تطبيقات Delphi، مازالت الـ client window والإطار موجودة، ولكن نادراً ما تستخدمها. ولجميع الأغراض العملية، يمكنك أن تعامل نافذة الإطار الـ client على أنهما نافذة واحدة، مقدمة لبرنامجك على أنها form النافذة الرئيسية. وتعتبر نوافذ الـ Document الصغيرة forms أيضاً، ولكن على عكس

نوافذ مثل الـ AboutBoxdialog التى تضيفها إلى الـ module، لا تستطيع نوافذ الـ MDI أن تجنح خارج حظيرتها . فهى مقيدة أن تظهر داخل الـ client area للنافذة الرئيسية . عندما تقوم بتصغير نافذة صغيرة، تعرض أيقونتها داخل النافذة الرئيسية، وليس فى الـ Windows Start bar .

وبإستثناء هذه الإختلافات، تعتبر نوافذ الـ MDI الصغيرة والكبيرة شبيهةبـ forms التطبيق ذات النافذة الواحدة فى تطبيقات الـ MDI، يمكنك إضافة toolbars، status line، جرافيك وcomponent objects، وتستخدم كل تقنيات البرمجة الأخرى لـ Delphi . ولكن تعتبر الـ MDI غاية فى التميز والإفادة عندما تستخدم فى تطبيقات تعمل بوثائق متعددة .

ملحوظة: تعتبر الـ MDI عادة نظام تعامل مع ملف، ولكن لا يجب على نوافذ التطبيق الصغيرة أن ترتبط بملفات قرص . يمكنك أيضاً إستخدام الـ MDI لإنشاء تطبيقات متعددة النوافذ من أنواع أخرى .

Note

والمصطلح «وثيقة» الذى يستخدم فى هذا الباب يشير إلى أى معلومات يمكن عرضها فى نافذة، وليس بالضرورة بيانات فى ملف قرص وثيقة .

النافذة الرئيسية للـ MDI:

إتبع هذه الخطوات لإنشاء النافذة الرئيسية لتطبيق الـ MDI :

١ - أبدأ تطبيقاً جديداً .

٢ - عين Name مثل الـ form MainForm .

٣ - حدد خاصية الـ FormStyle بـ fsMDIForm .

٤ - إحفظ المشروع فى دليل جديد . إجعل اسم الـ unit بـ Main.pas module، واسم MDITest إذا أردت الإستمرار) .

إن الـ form الرئيسية فقط هى التى يمكن أن تحدد خاصية الـ FormStyle بـ fsMDIForm ويمكن أن يكون هناك نافذة واحدة فقط مثل هذه لكل تطبيق . لضمان أن الـ form يتم إنشاؤه بصورة تلقائية، إختر الـ Project|Options، وتحقق من أن الـ MainForm موضحة فى Main form list ومذكورة تحت الـ forms Auto-create هذا هو كل ما تحتاج أن تفعله لإنشاء النافذة الرئيسية لتطبيق الـ

MDI . ويمكنك أن تنتقل الآن للفصل التـالى لإنشاء قـائمة برنامجك ونوافـذه الـ child .

ملحوظة: لتجربة التقنيات الموجودة فى هذا الباب ، أبدأ تطبيقاً جديداً بإستـخدام أمـر القـائمـة File|New Application ، أو إستـخـدم الـ File|New وإختر الـ Application فى مشروع MDI جديد مع قوائم إفتراضيـة ، و Child form ، event handlers مـؤخـراً فى هذا البـاب ، سـوف أوضح كيفية إستخدام مشروع الـ MDI ، ولكن لأغلب برامج العرض الموجودة فى هذا الباب يمكنك إنشاء تطبيق ذو نافذة واحدة خالية ثم أتبع الخطوات التالية .

إضافة Child MDI forms

يحتاج كل تطبيق MDI أيضاً إلى Child form واحدة على الأقل و unit .
إتبع هذه الخطوات لإضافة نافذة Child MDI إلى تطبيق MDI جديد :

١ - إبدأ تطبيقاً جديداً بإختيار أمر الـ File|New Application . قم بتغيير خاصية الـ Form1 Name ، بإستخدام الـ Object Inspector ، إلى MainForm . وقم بتغيير خاصية الـ MainForm FormStyle إلى fsMDIForm .

٢ - إختر File|Save All (أو إضغط زر السرعة Save All) . قم بتغـيير اسم الـ Unit1.pas إلى Main.pas وإحفظه . قم بتغـيـير الـ Project1.dpr إلى MDITest.dpr وإحفظه .

٣ - إختر File|New Form أو إضغط زر New form هذا يؤدى إلى إنشاء form جديد يسمى Form1 حسب النظام الإفتراضى . (إنه يسمى Form1 وليس Form2 لأنك قمت بتغيير اسم الـ form الأخرى لتصبح MainForm) . أعد تحديد حجم نافذة الـ Form1 لتجعلها أسهل فى الإختيار (إنما فى البداية تكون فى نفس حجم الـ MainForm وتغطى هذه النافذة فى الغالب) .

٤ - إستخدم الـ Object Inspector لتغيير خاصية الـ Name فى Form1 إلى ChildForm . حدد الـ FormStyle بـ fsMDIChild .

٥ - إختر File|Save All (أو إضغط زر Save All) . عليك إعطاء اسم ملف للـ ChildForm unit module . قم بتغـيـير هذا الأسم من Unit1.pas ،

وهو البديل الإفتراضي حسب النظام، إلى Child.pas، وإختر زر الـ Save.

٦- إختر الـ Project|Options لفتح الـ Options dialog. قم بإبراز الـ ChildForm المذكور فى اللوح الإيسر، وإضغط زر السهم الأيمن الأحادى الخط لنقل الـ object من قائمة الـ Auto-create forms إلى قائمة الـ Available forms. إذا لم تقم بهذه الخطوات، يتم إنشاء Child window بصورة تلقائية عندما يبدأ البرنامج.

لا يوجد شئ خطأ فى هذا من الناحية الفنية، ولكن النتيجة قد تحير المستخدمين الذين يتوقعون فتح وإنشاء نوافذ child بإستخدام أوامر مثل File|New أو File|Open. تقوم معظم تطبيقات الـ MDI بإنشاء نوافذها الـ child تحت تحكم البرنامج، وليس بصورة تلقائية عند تشغيل البرنامج، أغلق الـ Project Options dialog.

إن الخطوات السابقة تنشئ غلاف الـ MDI الهيكلى. يمكنك الآن إدخال البرمجة فى الغلاف لإقامة نوافذ child إستجابة لأوامر مثل الـ File|New والـ File|Open. يوضح الفصل التالى كيفية كتابة هذا الـ code.

Tip **فكرة:** إن النوافذ الـ child لتطبيق الـ MDI لا يجب أن تكون كلها من نفس النوع. إذا كان تطبيقك يحتاج أنواع مختلفة من النوافذ الـ child، إضافة أى عدد تحتاجه من الـ form الإضافية وذلك بتكرار الخطوات من ٣ إلى ٥. قم بإنشاء Name فريد لكل form جديدة، وحدده بـ fsMDIChild.

إحفظ المشروع بإستخدام أمر الـ File|Save All وإدخل أسماً فريداً لملف الـ pas. الخاص بالـ unit على سبيل المثال، يمكنك أن تسمى نوافذك الصغيرة بـ Child1Form وChild2Form وتحفظها جميعاً كملفات تسمى Child1.pas وChild2.pas. لمزيد من المعلومات عن إنشاء أنواع متعددة من forms النافذة الصغيرة، أنظر «العمل مع نوافذ صغيرة مختلفة النوع» فى هذا الباب.

فى هذه المرحلة، إذا كنت متابعاً، فمن بين المعلومات الأخرى فى دليل مشروعك يمكنك أن تجد الـ Pascal modules الثلاث هذه:

* Child.pas - هذه هى الـ Child form unit module. تحـتـــوى الـmodule على برمجة مخصصة لنوع الوثيقة أو معلومات أخرى متوفرة بواسطة Child window. وقد يكون أيضاً للنافذة الصغيرة قائمة، والتى يتم دمجها فى العادى فى قائمة الـ form الرئيسية.

* Main.pas هذه هـــى الـ Child form unit module، أدخـــل الـ event handlers فى هذه unit لأية object فى الـ form الرئيسية، وكذلك لبنود القائمة. ومن الناحية النموذجية، فهناك واحد على الأقل يجب أن ينشئ أمثلة للـ Child window للبرنامج. لمزيد من التفاصيل عن هذا الـ code، أنظر «Child window» و«إنشاء حالات Child window» لاحقاً فى هذا الباب.

* MDITest.dpr - هذا هو ملف مشروع البرنامج. أنه نادراً ما يحتاج إلى أية تعديلات لتطبيقات الـ MDI.

Tip **فكرة:** أصنع نسخة من دليل مشروعك الآن وإستخدم مشروع الـ MDITest.dpr لتجربة تقنيات البرمجة المتنوعة الموجودة فى هذا الباب.

إنشاء قائمة الـ MDI الرئيسية

إن كل تطبيق MDI يجب أن يكون له قائمة رئيسية. وهناك أمراً واحداً على الأقل فى هذه القـائمـة يجب أن ينشئ Child form object - وهذا هو أمـر الـ File|New. إتبع الخطوات التالية لإنشاء القائمة لتطبيق الـ MDITest الذى قمت بإنشائه فى الفصول السابقة. أفتح مشروع الـ MDITest.dpr، إذا لزم الأمر، ثم إتبع هذه الخطوات:

١- أضف MainMenu من لوحـــة الـ Standard فى الـ MainForm. (تأكد من إختيـار الـ MainForm، وليس الـ ChildForm). إضغط مرتين الـ MainForm لفتح الـ Menu Designer الخاص بـ Delphi.

٢- أكتب &File (علامة الـ & تشير إلى أن الحرف التالى، F، هو مفتاح القائمة) إضغط Enter ويقوم الـ Menu Designer بإنشاء قائمة File. إضغط File فى الـ Menu Designer - هذا يختـــار الـ TMenuItem (File1) فى الـ Object Inspector. قم بتغيير خاصية الـ Name للـ File1 لتصبح FileMenu.

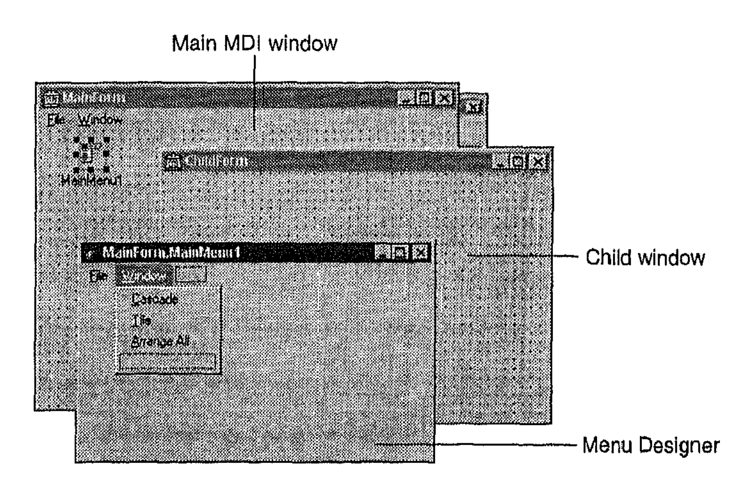
٣- مازلنا نستخدم الـ Menu Designer، أضف أوامر الـ &New، &Open...، و&Save... لقائمة الـ File . لتفعل هذا، إضغط تحت عنوان قائمة الـ File وأكتب الأوامر . كل أمر يتم إنشاؤه على أنه TMenuItem .

وفى هذا الباب، لقد إستخدمت أسماء الـ object الإفتراضية New1، Open1، وSave1، ولكن يمكن إختيار كل object بإستخدام الـ Menu Designer وتغيير خصائص الـ Name لهم فى نافذة الـ Object Inspector . للقائمة الآن أوامر File|New ، File|Open...، وFile|Save... وهو الحد الأدنى المطلوب من جانب تطبيقات الـ MDI . (إن الفصل «إنشاء Child window فى هذا الباب يشرح كيفية كتابة code لإنشاء Child window، وفتح وحفظ الوثائق، عندما يختار المستخدم أوامر القائمة هذه) .

٤- مازلنا نستخدم الـ Menu Designer، قم بإنشاء قائمة ثانية تسمى Windows . لتفعل هذا، إضغط بعد الـ File وأكتب &Window. إختر قائمة الـ Window (إضغطها فى Menu Designer) .

وبإستخدام نافذة الـ Object Inspector، قم بتغيير خاصية الـ Name للـ TMenuItem هذا من Window1، وهو البديل الإفتراضى له، إلى WindowMenu .

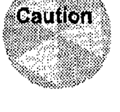
٥- نعود مرة أخرى إلى الـ Menu Designer، إضغط تحت الـ Window، وإدخل ثلاثة أوامر لهذه القائمة : &Cascade، &Tile، و&Arrange All كما هو الحال فى أوامر قائمة : &Cascade، &Title، و&Arrange All. كما هو الحال فى أوامر قائمة الـ File يمكنك إختيار كلاً من object الأوامر هذه وتغيير خصائص الـ Name لها، ولكن لبرنامج العرض هذا، إننا نستخدم أسماء object طبقاً للنظام الإفتراضية لـ Delphi وهى Cascade1، Tile1، Arrange1 . لتعرف كيف تكتب event handler لهذه الأوامر وأوامر أخرى لقائمة الـ Window، أنظر فصل «إستخدام أوامر قائمة Window» فى هذا الباب . يوضح شكل ١٥-١ الـ Menu Designer وقائمة الـ MDI التى تتطور فى هذه المرحلة .



شكل ١٥-١ الـ Menu Designer الخاص بـ Delphi
مع عناصر قائمة مقترحة لتطبيق الـ MDI .

٦- أغلق الـ Menu Designer أو إدفعه جانباً، وإختر الـ MainForm (إضغط F12 أو إستخدم الـ View|Project Manager لتجد الـ MainForm) . إستخدم الـ Object Inspector لتحدد خاصية الـ WindowMenu للـ MainForm بـ WindowMenu و TMenuItem . لتفعل هذا إضغط السهم الواقع بعد خاصية الـ WindowMenu وإختر الـ WindowMenu من القائمة الناتجة للـ object المتاحة للـ form . هذا التحديد يجعل التطبيق يذكر بصورة تلقائية عناوين الـ Child window المفتوحة فى القائمة المستهدفة - لا يجب عليك كتابة أية code لتجعل هذا يحدث .

بالطبع، إن أوامر القائمة الدقيقة فى تطبيقك تعتمد على إحتياجات برنامجك . ولكن فى الحد الأدنى، يجب أن يكون لتطبيق الـ MDI أوامر لإنشاء وفتح الـ Child window لوثيقة، وقائمة، وغالباً ما تسمى Window، لذكر عناوين النوافذ المفتوحة .



تحذير: يمكنك التعيين لخاصية الـ WindowMenu الـ top-level menu object فقط - بعبارة أخرى، TMenuItem الذى يمثل عنصر قائمة فى menu bar النافذة الرئيسية . لا تقم بتعيين أمر قائمة مثل الـ Open1 أو الـ Save1 لهذه الخاصية .

إن Child window form يمكن أيضاً أن يكون لها MainMenu . عندما يقوم المستخدم بتنشيط حالة للـ Child window، تقوم أوامر القائمة الخاصة بها بالإندماج بصورة تلقائية فى قائمة form أساسية طبقاً لخاصية الـ GroupIndex الخاصة بعناصر القائمة أنظر «القوائم المدمجة» فى هذا الباب لتعرف معلومات عن دمج قوائم الـ MDI .

الوصول للـ Child window

تقوم ثلاث خصائص للـ form بتوفير الوصول للـ Child window، وإنك تستخدمها فى أغلب تطبيقات الـ MDI . وخلال هذا الباب العديد من الأمثلة على كل من هذه الخصائص الثلاث :

- **ActiveMDIChild** - تعتبر إلى الـ Child window النشطة حالياً. إذا لم يوجد الـ Child window، تكون الـ ActiveMDIChild مساوية لـ nil . هذه الخاصية تعتبر مرجع للـ TForm، ولذلك، فإنك تحتاجها عادة لوضع الـ ActiveMDIChild فى الـ Child window class مثل الـ TChildForm .
- **MDIChildCount** – العدد الصحيح للـ Child window التى تمتلكها النوافذ الأم للـ MDI . إذا لم يوجد Child window، فإن الـ MDIChildCount تساوى صفر .
- **MDIChildren** – array للـ TForm لكل الـ Child window المملوكة لنافذة الـ MDI الرئيسية . والتعبير MDIChildren[0] يرجع إلى النافذة الصغيرة الأولى فى الـ array؛ والتعبير MDIChildren[MCIChildCount - 1] يعود على آخر Child window .

إذا كان الـ ActiveMDIChild يساوى nil، أو إذا كان الـ MDIChildCount يساوى صفر، فلا توجد Child window . فى مثل هذه الحالات، لا يجب أن يشير البرنامج إلى الـ MDIChildren array، بالرغم من أن فعل هذا لا يؤدى بالضرورة إلى توليد exception . وبالتحديد، يكون التعبير MDIChildren[0] مساوياً لـ nil عندما تكون الـ MDIChildCount تساوى صفراً .

الـChild window:

إن الـ Child window فى تطبيق الـ MDI تعـمل بطريقة مـشـابهـة للنافـذة الرئيـســيــة فى تطبــيق أحــادى النافــذة. الـ Child window يمكن أن يكون component objects مثل الـ Memos والـ Buttons. وتعرض الـ Child window وثيقة - على سبيل المثال، ملف نص أو bitmap. يمكن للمستخدمين أن يفتحوا Child window 1 متعددة، وترتب فى نظام متوال، وتصغر إلى أيقونات.

للـ Child window يمكن أن يكـون toolbars وStatus line، ولكن من الناحية التقليدية، مثل هذه الأنواع من الـ object تظهر عادة فى النافذة الرئيسية.

إن إختيار الـ Child window يمكن أيضاً أن يدمج أوامر فى القائمة الرئيسية للتطبيق. وهذه التقنية مفيدة بالأخص عندما تستخدم أنواعاً متعددة من Child window، كلاً بمتطلبات الأوامـر الخاصة بها. يوضح الفـصل التـالى تقنيات البرمـجة لهـذه الـ Child window ويذكـر غـلاف تطبيق MDI الذى يمكنك إستخدامه لتبدأ برنامجاً جديداً.

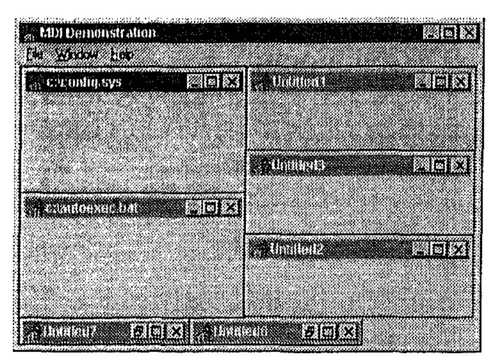
الـChild window ذات النوع واحد:



على القـرص المدمج: إن تطبيق الـ MDIDemo على القرص المدمج فى دليل الـ Source\MDIDemo يوضح كـيـفـيـة إنشـاء Child window ذات نوع احد.

يوضح البرنامج كيفية برمجة أوامر القائمة العامة مثل الـ File|New، الـ File|Open...، الـ Window|Cascade والـ Window|Tile. يوضح شكل ١٥-٢ عرض البرنامج مع بعض النوافذ المفتوحة.

توضح القائمة ١٥-١ الـ module الـ Child.pas لتطبيق الـ MDIDemo. غى هذا البرنامج، لا تعرض النافذة الصغيرة بيانات حقيقية، لذا فإن الـ module تكون بسيطة. ولكنهـا، توضح أدنى حد من الـ procedures يمكن أن توفـرها Child window.



شكل ١٥-٢: يوضح تطبيق الـ MDIDemo

برمجة Child window ذات نفس النوع.

القائمة ١٥-١: MDIDemo\Child.pas.

```
unit Child;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics,
   Controls,
    Forms, Dialogs, StdCtrls;

type
 TChildForm = class(TForm)
      procedure FormClose(Sender: TObject;
          var Action: TCloseAction);
          private
    { Private declarations }
       public
    { Public declarations }
        procedure LoadData(const FileName: String); virtual;
          procedure SaveData(const FileName:
      String); virtual;
       end;
```

```
var
    ChildForm: TChildForm;

implementation

{$R *.DFM}

procedure TChildForm.FormClose(Sender: TObject;
  var Action: TCloseAction);
begin
  Action := caFree;
end;

procedure TChildForm.LoadData(const FileName: String);
begin
    ShowMessage('LoadData from ' + FileName);
    Caption := LowerCase(FileName);
end;

procedure TChildForm.SaveData(const FileName: String);
begin
  ShowMessage('SaveData to ' + FileName);
    Caption := LowerCase(FileName);
end;

 end.
```

إن الـ Child window module يمكن أن توفـر procedure لتـحـمـيل وحـفظ بيـانات وثيـقـة. إن procedures مـثل الـ LoadData والـ SaveData يتم تعريفها كأعضاء عـامة من الـ TChildForm class. إن السبب فى جعلها عامة هو أن أوامر الـ Save والـ Open للـ modules الرئيسية يمكن أن تستدعيها.

وتقرر القائمة أيضاً الـ LoadData والـ SaveData الفعلية. وكنتيجة، يمكن أن ترث module أخـرى الـ TChildForm class وتغطى على الـ procedure. على سبيل المثال، يمكنك إضافة Child form للبرنامج وتأسيس الـ form class

على الـ TChildForm ، بدلاً من الـ TForm المعتادة . يمكنك عندئذ كتابة الـ LoadData procedures والـ SaveData لقراءة وكتابة البيانات الخاصة للـ module . إن وراثة class بهذه الطريقة غالباً ما يسمى subclassing a form (أو الـ class الفرعية لـ form) ، وهى تقنية موضحة فيما بعد فى هذا الباب .

فكرة: يجب عليك دائماً أن توفر OnClose لكل من الـ Child window . حدد الـ Action parameter بـ caFree حتى يترك التطبيق الـ Child window عندما تغلق نافذته . إذا لم تحدد الـ Action بـ caFree ، فان إغلاق النافذة يؤدى إلى تصغير حجمها فى الـ Child window client area بالنافذة الرئيسية .

بالاضافة إلى الـ OnClose ، يمكنك أيضاً أن تكتب OnCloseQuery لتحذر المستخدمين عندما يغلقوا نافذة بيانات غير محفوظة . فى الـ procedure ، حدد الـ OnClose بـ False لتمنع النافذة من الاغلاق ، حدد الـ CanClose بـ True إذا كان من الممكن اغلاق النافذة . انظر فصل "اغلاق نافذة ٨) فى الباب الثالث . لمزيد من المعلومات عن الـ OnClose والـ OnCloseQuery . إن النوافذ الرئيسية لتطبيق الـ MDI لا يمكن أن تغلق ، وبذلك ، لا يمكن أن ينتهى التطبيق ، إلا إذا كان من الممكن اغلاق جميع الـ Child window . وهذه الميزة الرائعة تساعد على منع فقد البيانات فى Child window مختفية وراء نافذة أخرى .

إن الـ LoadData procedures و SaveData للـ MDIDemo لا تقرأ أو تكتب أية ملفات حقيقية ، لذا فلك الحرية فى أن تلعب بأوامر البرنامج ...Open ، Save ، و Save As . للتأكيد أن البرنامج يستدعى الـ procedures فى الأوقات السليمة ، تعرض الـ ShowMessage أسم الملف ، تقوم LoadData procedures والـ SaveData بتعيين اسم الملف الحالى للـ Caption الخاص بـ Child form . وهذا يعرض اسم الملف ومساره فى address bar النافذة .

يمكنك ببساطة تعديل الـ MDIDemo ليقرأ ويكتب بيانات حقيقية . على سبيل المثال ، إدخل Memo فى ChildForm . (اذا لم تكن ترى هذه النافذة افتح MDIDemo.dpr ، اختر أمر الـ View|Project Manager ، اضغط علامة الزائد الواقعة بعد الـ Child ، واضغط مرتين الـ ChildForm) .

حـدد خـاصـيـة الـ Align للـ Memo1 الجـديد بـ alClient، والتى تجـعل الـ object يملأ نافـذته تماماً. اخـتـر خـاصـيـة Font مناسب إذا أردت، احـذف الـ "Memo1" من خاصية الـ Lines (اضغط الزر البيضاوى لفتح الـ editor)، وحدد الـ ScrollBars بـ ssBoth.

لإعادة برمجة ChildForm module، باستخدام الـ Project Manager، افتح ملف الـ Child.pas واختره فى code editor. وفى مكان عبارات الـ ShowMessage من القائمة (١٥-١)، ادخال الأوامر التالية لقراءة وكتابة ملفات النص.

```
{ Replace ShowMessage statement in LoadData with: }
Memo1.Lines.LoadFromFile(FileName);

{ Replace ShowMessage statement in SaveData with: }
Memo1.Lines.SaveToFile(FileName);
```

You have just constructed an MDI text editor!

لقد أنشأت الآن محور نص الـ MDI. اضغط F9 لتشغيل البرنامج المعدل. اختر أوامر الـ File|New لفتح Child window، والتى يمكنك أن تكتب داخلها نصاً أو تقوم بتغييرات. استخدم الـ File|Save لحفظ تغييراتك على قرص.

Caution

تحذير: ان الـ code السابق تجعل الـ MDIDemo تطبيقاً "حياً". والتغييرات التى تصنعها بالملف وتضغطها على القرص تكون دائمة. اذا كنت تلهو فقط، فاحفظ نسخ من أى ملفات تحررها.

ان اسماء المسارات الطويلة قد تبدو فوضوية فى الـ Captions الخاصة بالنوافذ، وقد تريد ان تستخدم الـ ExtractFileName function لتعيين اسماء ملف لـ address Bar بين الـ Child window بدون والـ drive المسار والـ drive. لتفعل هذا، اضف SysUtils إلى عبارة الـ Uses للـ module، وعرف متغير الـ String لتستخدمه فى عبارات المدخلات والمخرجات ولتعيينه لخاصية الـ Caption للنافذة نتيجة الـ ExtractFileName function من الـ SysUtils unit. انظر "نظم إدارة الملف" فى Online help لـ Delphi لمعرفة المزيد من المعلومات عنها وعن file function أخرى.

إنشاء Child window Instance:

على القرص المدمج : توضح القائمة ١٥-٢ تعريف الـ TMainForm class للـ MDIDemo و OnClick لقائمة الـ File . وهذه الجزئية من القائمة توضح كيفية إنشاء نماذج جديدة من الـ TChildForm class .
ولأن الـ module الرئيسية للـ MDIDemo طويلة ، فلذلك قمت بتقديمها فى أجزاء . لفحص القائمة الكاملة ، قم بتحميل مشروع الـ MDIDemo.dpr فى Delphi من على القرص المدمج فى دليل الـ Source\MDIDemo .

القائمة ١٥-٢: OnClick لقائمة File للـ MDIDemo.

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics,
    Controls,
      Forms, Dialogs, Menus, Child;

type
    TMainForm = class(TForm)
      MainMenu1: TMainMenu;
        FileMenu: TMenuItem;
        FileOpen: TMenuItem;
        FileSave: TMenuItem;
        FileSaveAs: TMenuItem;
        FileNew: TMenuItem;
        N1: TMenuItem;
        FileExit: TMenuItem;
        WindowMenu: TMenuItem;
        WindowCascade: TMenuItem;
        WindowTile: TMenuItem;
        WindowArrangeIcons: TMenuItem;
      N2: TMenuItem;
```

```
        WindowCloseAll: TMenuItem;
          WindowMinimizeAll: TMenuItem;
          HelpMenu: TMenuItem;
          HelpAbout: TMenuItem;
          OpenDialog: TOpenDialog;
          FileClose: TMenuItem;
          N3: TMenuItem;
          SaveDialog: TSaveDialog;
          procedure FileNewClick(Sender: TObject);
          procedure FileOpenClick(Sender: TObject);
          procedure FileCloseClick(Sender: TObject);
          procedure FileSaveClick(Sender: TObject);
          procedure FileSaveAsClick(Sender: TObject);
          procedure FileExitClick(Sender: TObject);
          procedure WindowCascadeClick(Sender:
        TObject);
          procedure WindowTileClick(Sender: TObject);
          procedure WindowArrangeIconsClick(Sender:
        TObject);
          procedure WindowMinimizeAllClick(Sender:
        TObject);
          procedure WindowCloseAllClick(Sender:
        TObject);
          procedure HelpAboutClick(Sender: TObject);
          procedure FileMenuClick(Sender: TObject);
          procedure WindowMenuClick(Sender: TObject);
      private
      {- Private declarations }
          procedure CreateChild(const Name: string);
            public
      {- Public declarations }
        end;
  var
   MainForm: TMainForm;

  implementation
```

```
{$R *.DFM}

const
 maxChildren = 10;  { Optional: No maximum required }

procedure TMainForm.CreateChild(const Name: String);
var
    Child: TChildForm;
begin
 Child := TChildForm.Create(Application);
    Child.Caption := Name;
end;

procedure TMainForm.FileNewClick(Sender: TObject);
begin
 CreateChild('Untitled' + IntToStr(MDIChildCount + 1));
end;

procedure TMainForm.FileOpenClick(Sender: TObject);
begin
 if OpenDialog.Execute then
    begin
     CreateChild(Lowercase(OpenDialog.FileName));
     with ActiveMDIChild as TChildForm do
        LoadData(OpenDialog.FileName);
    end;
end;

procedure TMainForm.FileCloseClick(Sender: TObject);
begin
 if ActiveMDIChild <> nil then
     ActiveMDIChild.Close;
end;

procedure TMainForm.FileSaveClick(Sender: TObject);
begin
```



```
    if Pos('Untitled', ActiveMDIChild.Caption) = 1 then
      FileSaveAsClick(Sender)
        else with ActiveMDIChild as TChildForm do
      SaveData(Caption);
  end;

  procedure TMainForm.FileSaveAsClick(Sender: TObject);
  var
   FExt: String;
  begin
   with SaveDialog do
      begin
       FileName := ActiveMDIChild.Caption;
          FExt := ExtractFileExt(FileName);
          if Length(FExt) = 0 then
            FExt := '.*';
              Filter := 'Files (*' + FExt + ')|*' + FExt;
          if Execute then
          with ActiveMDIChild as TChildForm do
            SaveData(FileName);
             end;
  end;

  procedure TMainForm.FileExitClick(Sender: TObject);
  begin
   Close;
  end;
```

إن الـ CreateChild procedure، والمقـــرر كـــعـــضـــو خـــاص فى الـ TMainForm class، ينشيء instances من الـ TChildForm. والبنود المقـــرر كعناصر خاصة تكون قابلة للوصول إليها فقط من methods فى الـ class. هذا يساعد على ضمان ان الـ modules الأخرى لا تقوم باستدعاء الـ procedures والـ functions والتى تؤدى خـدمـات هامـة بلا تمـيـيـز. ويوضـع الـ CreateChild procedure ايضاً كيفية إنشاء form فى وقت التشغيل. لتفعل هذا، قم بتعريف متغير من نوع الـ class :

```
var
   Child: TChildForm;
```

بعد ذلك ، وفى هيكل الـ procedure، قم باستدعاء الـ Create methods للـ class مع الـ Application كـ argument قم بتعيين الـ object الناتج إلى المتغير :

```
   Child := TChildForm.Create(Application);
```

إن السبب فى تمرير الـ Application ، وليس الـ MainForm ، إلى الـ Create هو أن الـ Application يمثل نافذة الإطار للـ MDI، وهى النافذة الحقيقية التى تملك الـ Child window . والـ MainForm ما هى إلا أمر بديلة لصغارها .

إن FileNew event handler هو ، الـ FileNewClick procedure ، يستدعى الـ CreateChild . و FileOpen يقوم بنفس الشئ، ولكن أيضاً يستدعى procedure العام LoadData للـ Child.pas . ولكن، وما هذه إلا تقنيات مقترحة لإقامة Child window وتحميل بيانات من على ملفات؛ فأنت حر فى كتابة الـ module كما تشاء .

إستخدم خاصية الـ ActiveMDIChild للـ MainForm لأداء عمليات على الـ Child window الحالية . وهذه الخاصية تساوى nil إذا لم يوجد Child window، ويجب أن تفحص هذا دائماً قبل أن تستخدم الخاصية . على سبيل المثال، إن الـ FileClose event وهى الـ FileCloseClick procedure، تغلق Child window النشطة بعبارة :

```
if ActiveMDIChild <> nil then
   ActiveMDIChild.Close;
```

لا تستخدم أبداً عبارة غير مقيدة مثل التالية، والتى تولد exception إذا لم يوجد Child window.

هذا يحدث لأن العبارة قد تحاول أن تستدعى الـ Close) method) كـ nil reference :

```
   ActiveMDIChild.Close; /// ???
```



كما إقترحت، يجب أن نتأكد من أن الـ Child window module تحفظ البيانات الخاصة بها فى OnClose و OnCloseQuery. لا تغلق النافذة فعلياً إلا إذا كانت الـ Child object تحققت من أنها يمكنها أن تغلق.

إن الـ procedures FileSaveClick والـ FileSaveAsClick لـ MainForm تقوم بإستدعاء الـ SaveData procedure لل Child window النشط. إذا كان Caption النافذة Untitled، فإن الـ FileSaveClick هو الإفتراضى لـ File save-as click؛ وإلا، تقوم الـ code بإستدعاء الـ SaveData مع اسم الملف الحالى، والمأخوذ من Caption النافذة. ويستخدم الـ FileSaveAsClick التقنية المقترحة فى الباب الثانى عشر لقصر ملفات الـ SaveDialog على أولئك الذين لهم نفس الإمتداد كإمتداد النافذة الحالية.

وأخيراً، فى هذه الجزئية من القائمة، إن FileExit event handler لل MainForm، FileExitClick، يستدعى الـ Close.

إن بإمكانه أن يفعل هذا لأن الـ object الأم يحاول أن يغلق ويحرر كل الـ Child window. ولكن، ينتهى التطبيق فقط إذا كان من الممكن إغلاق كل الـ Child window.

إستخدام أوامر قائمة الـ Window

إن أغلب تطبيقات الـ MDI توفر قائمة Window، والتى تؤدى عمليات على الـ Child window مثل التتالى، التجانس، والترتيب للأيقونات فى الـ client area بالنافذة الرئيسية. توضح القائمة ١٥-٣ OnClick لقائمة الـ Window لل MDIDemo وتظهر كيفية كتابة أوامر نافذة مخصصة مثل الـ Close all، التى لا يقدمه الـ Delphi.

القائمة ١٥-٣: OnClick لقائمة الـ Window لل MDIDemo.

```
procedure TMainForm.WindowCascadeClick(Sender: TObject);
begin
  Cascade;
   end;

procedure TMainForm.WindowTileClick(Sender: TObject);
```

```
begin
  Tile;
end;

procedure TMainForm.WindowArrangeIconsClick(Sender: TObject);
begin
    ArrangeIcons;
end;

procedure TMainForm.WindowMinimizeAllClick(Sender:
TObject);
var
  I: Integer;
begin
  for I := MDIChildCount - 1 downto 0 do
     MDIChildren[I].WindowState := wsMinimized;
end;

procedure TMainForm.WindowCloseAllClick(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to MDIChildCount - 1 do
     MDIChildren[I].Close;
end;
```

إن الـ procedures الثلاثة الأولى تنفذ أوامر قائمة الـ Window المعيارية : Cascade ، Tile ، Arrange الأيقونات . ولأن هذه الأوامر عامة جداً، فإن الـ TForm class توفر الـ code اللازمة فى الـ Cascade method والـ Tile والـ ArrangeIcons كإستجابة لأوامر القائمة المناسبة . يمكنك أيضاً إستدعاء هذه الـ method فى أوقات أخرى - على سبيل المثال، إستجابة للـ OnClick الخاص بالـ SpeedButton فى toolbar .

هناك أيضاً أثنين من الأوامر المعيارية التى يمكنك أن تضيفها لقائمة الـ Window هما الـ Next والـ Previous . ببساطة إستدع الـ Previous methods

والـ Next للـ form الرئيسية. ولكن، قد تريد إبطال هذه الأوامر إذا كانت نافذة واحدة فقط تستخدم عبارات مثل التالية فى OnClick الخاص بقائمة الـ Window (بفرض الـ WindowNext والـ WindowPrevious، TMenuItem objects)

```
WindowNext.Enabled := MDIChildCount > 1;
WindowPrevious.Enabled := WindowNext.Enabled;
```

بالإضافة للك وأمر المعيارية لقائمة الـ Window يمكنك كتابة أوامر جديدة لأداء عمليات على كل الـ Child window. على سبيل المثال، يقوم الـ MDIDemo بتنفيذ أمر الـ Window|Minimize all مع الـ for loop هذه :

```
for I := MDIChildCount - 1 downto 0 do
  MDIChildren[I].WindowState := wsMinimized;
```

لتجعل الأيقونة تخرج فى ترتيب معقول، تقوم الـ for loop بالعد من آخر Child window إلى الأولى. إستخدم array ،MDIChildren لـ form النافذة الرئيسية للوصول إلى كل Child window. وعناصر الـ array من نوع الـ TForm، لذا فإن لم تكن تستخدم خاصية أو method مورث مثل الـ WindowState الموضح هذا، فإنك تحتاج أن تكتب تعبيراً لإستدعاء methods وإستخدام خصائص الـ Child window class الخاصة بك. على سبيل المثال، العبارات التالية تستدعى الـ YourMethod procedure للـ Child window الأولى :

```
if MDIChildCount > 0 then
  TChildForm(MDIChildren[0]).YourMethod;
```

وبالتبادل إستخدم عبارة with مثل هذه :

```
if MDIChildCount > 0 then
with MDIChildren[0] as TChildForm do
 YourMethod;
```

إن تطبيق الـ MDI المصمم بشكل سليم يجب أن يبحث دائماً عن الـ Child window واحدة على الأقل قبل أداء أعمال من خلال الـ MDIChildren. أما أن يبحث ما إذا كان الـ MDIChildCount أكبر من صفر، أو يختبر ما إذا كان الـ ActiveMDIChild لا يساوى nil.

بالرغم من أن ليس كل تطبيقات الـ MDI توفر أمر الـ WindowlClose all ، ربما كان يجب أن يكونوا كذلك . إننى أحاول أن أفتح العديد من النوافذ وأنا أعمل على مختلف المشروعات ومن المفيد أن يكون لديك method لإغلاق جميع النوافذ دون أن تضطر إلى إغلاقها واحدة واحدة . يؤدى الـ MIDDemo هذا العمل فى الـ procedure ، WindowCloseAllClick ، والذى ينفذ الـ for loop لإغلاق النوافذ بالترتيب من الأولى إلى الأخيرة :

```
for I := MDIChildCount - 1 downto 0 do
  MDIChildren[I].Close;
```

داخلياً ، يتم تمثيل الـ MDIChildren كـ TList object ، ولذلك ، فمن المسموح أن تؤدى عمليات مثل السابقة التى تؤثر على عدد الـ Child window .

مجموعة من أوامر الـ MDI

توضح القائمة ١٥-٤ باقى الـ source code ، Main.pas للـ MDIDemo .

القائمة ١٥-٤: procedures متنوعة للـ MDIDemo.

```
procedure TMainForm.HelpAboutClick(Sender: TObject);
begin
  AboutForm.ShowModal;
end;

procedure TMainForm.FileMenuClick(Sender: TObject);
begin
  FileNew.Enabled := MDIChildCount < maxChildren;
    FileOpen.Enabled := FileNew.Enabled;
    FileClose.Enabled := MDIChildCount > 0;
    FileSave.Enabled := FileClose.Enabled;
    FileSaveAs.Enabled := FileClose.Enabled;
end;

procedure TMainForm.WindowMenuClick(Sender: TObject);
var
```



```
    I: Integer;
  begin
    with WindowMenu do
      for I := 0 to Count - 1 do
        with Items[I] as TMenuItem do
            Enabled := MDIChildCount > 0;
  end;

  end.
```

إن الـ HelpAboutClick procedure يعــرض الـ AboutBox dialog الخاص بالبرنامج، وهو غير موضح هنا. إن الـ FileMenuClick Procedures والــ WindowMenuClick هــى OnClick لــلـــ FileMenu والـــ WindowMenu. يستدعى البرنامج هذه الـ procedures عندما يفتح المستخدم القوائم تقوم الـ procedures بإبطال وتشغيل الأوامر على أساس ظروف البرنامج. على سـبيل المثـال يتم إبطال الـ File|New عندما يكون الـ MDIChildCount مساوياً لـ maxChildren، والذى يحد من عدد الـ Child window التى يفتحها المستخدم. إن كل أوامر قائمة الـ Window يتم إبطالها عندما لا يوجد Child window. إذا أضفت أوامر Next وPrevious كما هو مقترح، يجب أن تقوم بتشغيلها منفصلة إذا كان الـ MDIChildCount أكبر من واحد.

ملحوظة: إنك لا تحتاج أن تحد من عدد الـ Child window. لقد ضمنت هذه الميزة فى الـ MDIDemo فقط لإظهار البرمجة. يمكن لتطبيق الـ MDI أن يكون له أى عدد من الـ Child window بقدر ما تسمح الذاكرة والـ Resource الأخرى.

العمل مع الـ Child window مختلفة النوع



على القرص المدمج: إن الـ Child window لتطبيق الـ MDI قد تكون من أنواع مختلفة. لكل نوع من الـ Child window، أضف form جديدة للمشروع وأضف برمجة للـ module الرئيسية لإنشاء instances نافذة. على سبيل المثال، قم بتجربة الخطوات التالية لتشغيل للـ

Child window MDIDemo تستطيع عرض ملفات bitmap . (يوجد البرنامج التام على القرص المدمج فى دليل Source\MDIDemo2 . إذا لم تكن تريد القيام بهذه التعديلات بنفسك ، أفتح ملف مشروع MDIDemo2.dpr فى هذا الدليل لترى الملفات التامة مع Delphi . هذه الخطوات تفترض أنك تقوم بتعديل المشروع الأصلى فى الـ Source\MDIDemo) .

١ - أفتح ملف مشروع MDIDemo.dpr . أضف module أخرى للـ MDIDemo بإختيار الـ File|New Form أو بضغط الـ New Form SpeedButton. .

٢- قم بتغيير خاصية الـ Name للـ form الجديدة Form1 إلى ChildBmpForm . حدد الـ FormStyle بـ fsMDIChild .

أختر Project|Options وأنقل الـ ChildBmpForm من قائمة الـ Auto-create forms إلى قائمة الـ Available forms يجب أن تذكر أن الـ panel اليسرى الـ MainForm والـ AboutForm . أغلق الـ Project Options dialog بضغط OK .

٣- إحفظ المشروع بإختيار File|Save All . عندما يطلب منك اسم ملف ، إدخل Childbmp.pas بدلاً من الأسم الإفتراضى ، Unit1.pas .

٤ - أضف الـ ChildBmp form بالضغط داخل نافذتها . إنتقل إلى الـ Object Inspector ، إضغط Events ، وإضغط مرتين المسافة الواقعة إلى اليمين من الـ OnClose لإنشاء event handler لهذا الـ event . أكتب عبارة تعين caFree للـ Action . وهذا يمحو الـ form من الذاكرة عندما تغلق الـ Child window . إذا لم تقم بهذه الخطوة فإن إغلاق النافذة يؤدى إلى تصغيرها فى الـ client area للنافذة الرئيسية - هذا مسموح ، ولكنه فى الغالب ليس هو ما تريد . يجب أن يبدو OnClose مثل هذا :

```
procedure TChildBmpForm.FormClose(Sender: TObject;
  var Action: TCloseAction);
begin
  Action := caFree;
end;
```



٥- أضف Image من Additional palette فى الـ ChildBmp form.
حدد خاصية الـ Align للـ Image1 بـ alClient حتى تملأ النافذة وحدد خاصية الـ Stretch Image1 بـ Ture. هذه التغييرات تملأ الـ Child window بالصورة التى تم تحميلها فى وقت التشغيل وتحدد حجم الصورة لتتناسب مع الـ client area للـ Child window.

٦- إنتقل للـ code editor وأختر ChildBmp unit. أضف Child فى عبارة الـ uses فى أعلى الـ module. هذا ضروريا لأننا نحتاج أن ترث class من الـ Child form الأصلية، إن عبارة الـ uses التامة يجب أن تبدو مثل هذه:

```
uses
   Windows, Messages, SysUtils, Classes, Graphics, Controls,
    Forms, Dialogs, Child;
```

٧- بينما لازلنا فى الـ code editor للـ ChildBmp قم بتغيير الـ class الأساسية للـ TChildBmpForm من TForm إلى TChildForm هذا يجعل الـ TChildBmpForm ترث ٨ خصائص و methods من الـ TChildForm، وهذه تقنية تسمى الـ subclassing للـ form إن تعريف الـ class المعدلة للـ TChildBmpForm يجب أن يبدو الآن مثل هذا:

```
type
  TChildBmpForm = class(TChildForm)
    ...
```

٨- أضف الـ LoadData procedures والـ SaveData لـ TChildBmpForm class. قم بتصميم الـ procedures مع الـ override directive، الذى يخبر Delphi أن يستبدل الـ methods الفعلية الموروثة من الـ TChildForm. إدخل هذه التعاريف تحت الكلمة الرئيسية public فى تعريف الـ class TChildBmpForm:

```
public
  procedure LoadData(const FileName: String); override;
    procedure SaveData(const FileName: String); override;
```



٩- قم بتنفيذ الـprocedure في قطاع الـ implementation للـChildBmp module . توضح القائمة ١٥-٥ الـ code التامة ، التى يمكنك إستخدامها كمرشد لك لإتمام برنامجك .

هذه القائمة مع القوائم الأخرى فى هذا الفصل ، موجودة فى دليل الـ Source\MDIDemo2 على القرص المدمج .

١٠- أختر الـ ChildBmpForm فى الـ Object Inspector ، إضغط الـ Events page tab ، وإضغط مرتين المسافة الواقعة إلى اليمين من الـ OnCreate event . أدخل الـ code من الـ FormCreate procedure بالقائمة ١٥-٥ . هذا يؤدى إلى تحرير الـ Memo1 الموروث والذى قد يتنازع مع الـ Image1 فى الـ class الجديدة إذا لم يتم تحريره . (القائمة ١٥-٥ والقوائم الأخرى فى هذا الفصل توجد على القرص المدمج فى دليل Source\MDIDemo2 .

القائمة ١٥-٥: الـ source code للـ ChildBmp module التامة.

```
unit Childbmp;

interface

uses
    Windows, Messages, SysUtils, Classes, Graphics,
  Controls,
    Forms, Dialogs, Child, ExtCtrls;

type
 TChildBmpForm = class(TChildForm)
     Image1: TImage;
       procedure FormClose(Sender: TObject;
         var Action: TCloseAction);
           procedure FormCreate(Sender: TObject);
      private
        { Private declarations }
    public
     { Public declarations }
       procedure LoadData(const FileName: String);
```



```
      override;
        procedure SaveData(const FileName: String);
      override;
       end;

    ChildBmpForm: TChildBmpForm;

implementation

{$R *.DFM}

procedure TChildBmpForm.FormClose(Sender: TObject;
 var Action: TCloseAction);
begin
 Action := caFree;
end;

procedure TChildBmpForm.LoadData(const FileName: String);
begin
 Image1.Picture.LoadFromFile(FileName);
    Caption := LowerCase(FileName);
end;

procedure TChildBmpForm.SaveData(const FileName: String);
begin
    Image1.Picture.SaveToFile(FileName);
    Caption := LowerCase(FileName);
end;

procedure TChildBmpForm.FormCreate(Sender: TObject);
begin
 inherited;
    Memo1.Free;
end;

end.
```



Note

ملحوظة: إن السبب فى تحرير الـ Memo1 فى الـ FormCreate method هو منع هذا الـ object من التضارب مع الـ Image1 فى الـ TChildBmpForm class . هذا يُظاهر إحدى المشاكل الاساسية مع الـ classes الفرعية للنوافذ- قد تضيف الـ class الجديدة objects مع objects فى الـ class السالفة لها . انظر مشروع ١٥-٦ من المقترحات عن كيفية تحسين ميزات الـ class الخاص بالبرنامج .

إن الخطوة الأخيرة فى تعديل الـ MDIDemo لقراءة وكتابة ملفات bitmap هو إعادة برمجة الـMain module بحيث تستطيع أن تنشئ حالات من الـ bitmap class الصغيرة . لتفعل هذا، اختر الـ Main module فى محرر الـ code (استخدم الـ View|Project Manager لفتح الـ MainForm إذا كانت الـ Main module غير مرئية) . أضف ChildBmp إلى عبارة الـ Uses للـ Main module، والتى يجب أن تكون مثل هذه :

```
uses
Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs, Menus, Child, ChildBmp;
```

On the CD-ROM

على القرص المدمج : اعد كتابة الـ CreateChild procedure كما هو موضح فى القائمة ١٥-٦ . (يوجد هذا النص فى دليل الـ Source\MDIDemo2 على القرص المدمج .) اضغط F9 ولتشغيل البرنامج المعدل . افتح ملف bitmap . (إذا أضفت Memo إلى الـ Child module كما هو مقترح، يمكنك أيضاً فتح ملفات نص .) يقوم البرنامج بانشاء النوع المناسب من النوافذ على اساس امتدد اسم الملف .

القائمة ١٥-٦: الـ CreateChild procedure الذى تم إعادة برمجته، والخاص بالـ Main module، لتطبيق الـ MDIDemo المعدل، وكذلك أضف ChildBmp لعبارة الـ Uses للـ module.

```
procedure TMainForm.CreateChild(const Name: String);
var
  Child: TChildForm;
    FExt: String;
begin
```




```
      FExt := ExtractFileExt(Name);
       if FExt = '.bmp' then
        Child := TChildBmpForm.Create(Application)
         else
        Child := TChildForm.Create(Application);
         Child.Caption := Name;
  end;
```

فى الـ CreateChild procedure الجـديد، إذا كـان bmp . هو إمـتـداد الملف، ينشئ البـرنامج TChildBmpForm Child window، وإن لـم يكن bmp . ، فـإن البـرنامج ينشئ TChildForm . ولأن الـ LoadData والـ SaveData procedures تعتبر فعلية (راجع القائمة ١٥-١ و١٥-٥)، فإن نوع Child window object يحدد نوع البيانات التى يتم تحميلها.

وحتى تفـهم جيداً قيمـة استخدام methods فعلية فى الـ forms من نوع الـ class الفرعية، انظر الـ FileOpenClick procedure الخاص بالـ Main module فى القـائمـة ١٥-٢ . إذا كـانت Child window هى TChildForm، يقـوم procedure باستـدعـاء الـ LoadData procedure الأصلى . إذا كـانت النافـذة هى TChildBmpForm، يقوم الـ procedure باستدعاء الـ method الذى تم تخطيطه. وهذه الاستـدعـاءات لا يتم توجيـهـها بواسطة الـ code الظاهر فى البرنامج، ولكن بواسطة أنواع Child window المشار إليها عن طريق الـ ActiveMDIChild.

وبسبب تصميم البرنامج المختص بالـ objects، فمن السهل نسبياً إضافة أنواع أخرى من Child window . ببساطة، قم بإنشاء وبرمجة form module جديدة كما فعلت مع الـ TChildBmpForm . استخلص الـ form class الجديدة من الـ TChildForm واكـتـب الـ LoadData procedures والـ SaveData . قم بتـعـديل الـCreateChild فى الـ Main module لإنشـاء حـالات من الـclass الجديدة، وبذلك تكون قد اتممت العمل.

دمج القوائم:

فى تطبـيـقـات الـ MDI وبالأخص مع الأنواع المتـعـددة من الـ Child window، قد تريد تعديل أوامر القائمة أو حتى إضافة قوائم، إعتماداً على أى نوع من الـ Child window تكون نشطة. هذا يسهل فعله، ولكن يتطلب الانتباه إلى بعض التفاصيل التى قد لا تكون واضحة.



فى النافذة الأم- تلك التى بها الـ FormStyle محددة بـ fsMDIForm- يجب أن يحتوى الـ MainMenu على أوامر عامة تنطبق بشكل عام على كل الـ Child window على سبيل المثال، توفر قائمة النافذة الأم عادة قائمة Window ذات أوامر Cascade ، Tile وغيرها التى تؤثر على كل النوافذ بغض النظر عن نوعها . ويجب أيضاً أن يكون لقائمة النافذة الرئيسية أوامر مثل الـ File|New ، File|Open، File|Close... التى تفتح وتغلق النوافذ الصغيرة .

فى كل نافذة صغيرة تتطلب أوامر فريدة، أو تحتاج إلى تغيير بنود القائمة بناء على ظروف متنوعة، يمكنك إدخال MainMenu آخر . إننى أحب أن أسمى هذه الـ objects طبقاً للـ module- على سبيل المثال، ChildFormMenu أو ChildBmpFormMenu . إدخل أوامر فى الـ MainMenu هذه للدمج مع قائمة النافذة الأم .

إن دمج القوائم فى تطبيقات الـ MDI يكون تلقائياً لأى الـ Child window لها MainMenu . لهذا السبب، يجب أن تحدد كل خصائص الـ AutoMerge للـ MainMenu بـ False . ومن الناحية الفنية، يجب أن تكون خاصية الـ AutoMerge للنافذة الأم فقط هى التى تحدد بـ False. يتم تجاهل قيم الـ AutoMerge للـ MainMenu للـ Child window، ولكن حددها جميعاً بـ False على أى حال .

انظر الباب الخامس لمعرفة التعليمات عن دمج القوائم فى تطبيقات غير الـ MDI . التطبيقات غير الـ MDI فقط هى التى تستخدم خاصية الـ AutoMerge .

بعد ذلك، قم بتعيين قيم GroupIndex إلى عناصر قائمة معينة . تحدد هذه القيم مكان إدخال القوائم وما إذا كنت سوف تستبدل عنصر قائمة أو تضيف أوامر جديدة . عند الدمج، يستخدم التطبيق قيم الـ GroupIndex طبقاً للقواعد التالية :

* يتم استبدال بنود القائمة التى لها نفس قيم الـ GroupIndex . على سبيل المثال، إذا كان عنصر MainMenu له GroupIndex مساوياً لعشرة، فإن أى عناصر قائمة فى MainMenu خاصة بـ Child window لها نفس الـ GroupIndex تحل محل البنود فى النافذة الأم .

* يتم إدخال عناصر القائمة ذات الـ GroupIndex الفريدة . قم بتعيين قيم أعلى لإدخال عناصر قائمة فى الـ Child window إلى اليمين من البنود ذات القيم المنخفضة لإدخال عناصر قائمة إلى اليسار من العناصر فى النافذة الأم .

بالإضـافـة إلى دمج القـوائم، يمكنك أيضـاً اسـتـدعـاء الـ TMenuItem methods لتشغيل وابطال عناصر القائمة، لإدخال أو تغيير أوامر، ولأداء حيل أخرى للقائمة. على سبيل المثال، تستطيع الـ Child window إضافة علامة صح لأمر الـ Options|Save بعبارة مثل هذه:

```
MainForm.OptionsSave.Checked := True;
```

والحيلة هنا هى التمكن من الوصـول إلى الـ Child module. افعل هذا بإضافة Main (أو أى اسم unit اخرى) لعبارة الـ Uses فى الـ Child module. لمنع الـ reference الـ cride، يجب أن تفعل هذا فى تنفيـذ الـ Child window، وليس فى عبارة الـ Uses فى الجزء الـ interface. ولأن عبارة الـ Uses للـ Main module تشير بالفعل للـ Child module، فإن الإشارة مرة أخرى للـ Main فى الـ Child window تؤدى إلى الـ reference المعادة، والذى تسمح به الـ Object Pascal. فى الـ Child module، قم بوضع كلمة التنفيذ الرئيسية وأمر تحميل الـ form، وأضف عبارة Uses كما هو موضح هنا. يمكنك عندئذ الوصـول إلى عناصر قائمة فى الـ MainForm:

```
implementation
{$R *.DFM}
uses Main;
```

وتحدث الـ unit references الـ Cricle عندما تقوم أثنان من modules - ولنسميهما Chicken وEgg- بالإشارة إلى بعضهما فى عبارات الـ Uses لقطاعات واجهة التطبيق الخاصة بهما. وتسمح الـ Object Pascal بهذا بسبب امكانية أن تستخدم الـ Chicken module كتعريف فى الـ Egg module، والذى قد يتطلب تعريفاً من الـ Chicken module. لحل تناقض الـ Chicken-and-Egg، اضف تركيبة ثانوية إلى عبارة Uses فى القطاع interface للـ module الأساسية، اضف التركيبة الأساسية إلى عبارة Uses فى قطاع الـ implementation للتركيبة الثانوية. ويمكن ألا تشير Unit interfaces إلى بعضها والبعض (circularly)، ولكن قطاعات الـ implementation الخاصة بها قد تفعل هذا وبلا حدود.

تقنيات MDI أخرى:

أثناء كتابة هذا الباب، مررت بكثير من أجزاء الـ MDI ضمن نماذج الـ source code التى شرحتها. فيما يلى بعض تقنيات الـ MDI الأخرى التى قد تجدها نافعة.

الوصول إلى نوافذ الإطار والـ client:

قد يكون ضرورياً فى بعض الأوقات أن تتمكن من الوصول إلى نوافذ الـ client والإطار لتطبيقات الـ MDI. إنك لن تحتاج غالباً إلى استخدام هذه التقنيات إلا إذا استدعيت الـ Windows API التى تتطلب window handles. يجب أن توفر الـ TForm class الخاصة بـ Delphi خدمات كافية لأغلب تطبيقات الـ MDI.

استخدم خاصية الـ ClientHandle فى الـ form الرئيسية للإشارة إلى نافذة الـ client. فى البرمجة الكلاسيكية للـ MDI و توفر هذه النافذة عمليات عامة- مثل التحكم فى والوسائل- التى تنطبق على كل الـ Child window.

Note

ملحوظة: إن قيمة الـ ClientHandle، من نوع الـ HWnd (window handle)، تكون صالحة قط فى forms ذات FormStyle محددة بـ fsMDIForm.

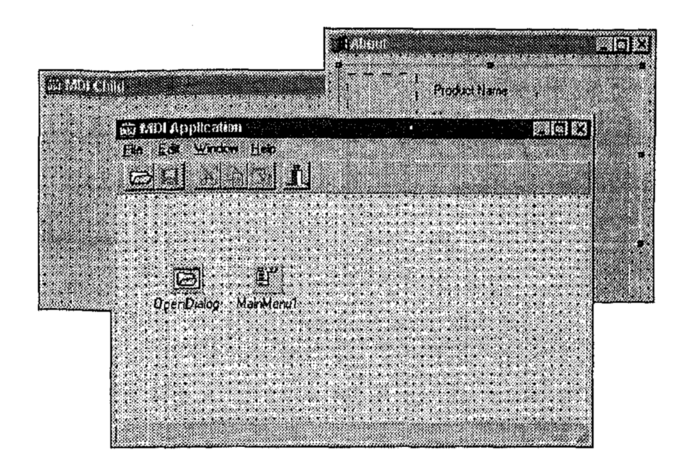
يمكنك الإشارة إلى نافذة الإطار باستخدام خاصية الـ Handle للـ form الرئيسية. وهذه القيمة دائماً تذكر Window Handel للـ form فى كلا من تطبيقات الـ MDI وغيرها. وفى form ذات FormStyle مساوية لـ fsMDIForm، تشير خاصية الـ Handle إلى نافذة إطار الـ MDI.

استخدام الـ MDI Application Template:

للبرمجة السريعة لأنماط الـ MDI النموذجية، استخدم الـ MDI Application Template الخاصة بـ Delphi. أولاً، قم بإنشاء دليلاً خالياً لتخزين ملفات المشروع. بعد ذلك، اختر File|New...، اضغط الـ Projects page tab فى نافذة الـ New Items dialog، واختر MDI Application. اضغط OK ثم فى dialog إختيار الدليل الجديد حيث تريد تخزين ملفات المشروع.



أغلق نافـذة dialog الدليل ، ويقـوم Delphi بإنشـاء تـطبيق MDI له status panel ، toolbar وقـائمـة ذات بنود File ، Edit ، Window ، وHelp . يوضـح شكل ١٥-٣ نافـذة الـ form الناتجـة إلى أعلى نـوافـذ الـ Child والـ About اللذان ينشئهما الـ template أيضاً .



شكل ١٥-٣: تنشئ الـ MDI Application Template form لها status panel ، toolbar وقائمة رئيسية. وتنشئ أيضاً نوافذ Child وAbout والظاهرة هنا مختفية جزئياً خلف الـ form الرئيسية.

الإرشادات التالية تساعدك على البدء فى إستخدام الـ MDI Application Template :

* الـ ChildWin.pas module تعـرف الـ TMDIChild يمكنك إضـافـة الـ LoadData procedure والـ SaveData فى هذه الـ class ، أو يمكنك قراءة وكتابة بيـانات والـ Main module . إذا كـان تطـبيـقـه له نوع Child window واحـدة ، يمكنك إضافة البرمجة إلى الـ Main ، وإن لم يكن ، إجعل كل class نافذة مسئولة عن مدخلاتها ومخرجاتها ، وأسس calsses نافذة أخرى على الـ TMDIChild .

* حـسب النظام الإفـتـراضى للبـرنامج ، يتم تسـمـيـة الـ Application بـ MDIApp . لتـعـيـد تسـمـيـتـه ، أخـتـر File|Save Project ، وإدخل اسم ملف جديد . قم بحفظ وإجراء عملية الـ compile للمشروع . يمكنك عندئذ حـذف كل ملفات الـ *.MDIApp من دليل المشروع .



* الـ Main.pas module تعـرف الـ TMainForm class . ملحـوظة : راجع تعريفات الـ class في الـ module للحصول على أسماء object متنوعة . بعض الأسماء، مثل Edit1، تكون إفتراضية طبقاً للبرنامج، والبعض الآخر، مثل SaveBtn، ويتم تسميتها طبقاً للإستخدام . إن تقاليد التسمية الخاصة بالـ template ليست كما تسمى التطبيقات في هذا الكتاب . يمكنك تغييرها، ولكن إذا فعلت هذا، يجب أن تغييرها في كل مكان تحدث به . فليس من الكافي، مثلاً، أن تغيير خاصية الـ Name كـ object - فيجب عليك أيضاً أن تبحث وتستبدل بصفه عامة اسم الـ object في أي عبارات تشير إليه . يقوم Delphi بصورة تلقائية بتحديث تعريفات اسم الـ object فقط، فهو لا يبحث عنه ويستبدل أسماء الـ object في العبارات .

* أختر الـ View|Project Manager وقم بإبراز الـ ChildWin لفتح الـ form و Child window unit . يمكنك عندئذ إدخال أية objects تريدها في الـ Child window، إنشاء قائمة مدمجة، وكتابة code لأوامر الـ Child window .

* في الـ Main module، أضف عبارات الـ data-loading للـ FileOpenItemClick procedure . كما ذكرت، يمكنك قراءة ملفات مباشرة، أو يمكنك إستدعاء الـ LoadData procedure لنافذة ActiveMDIChild .

* أضف SaveDialog على الـ form الرئيسية، وأكتب code لحفظ بيانات في الـ FileSaveItemClick . أو، يمكنك إستدعاء الـ SaveData procedure الذي أضفته لـ class النافذة الصغيرة .

* إختيارياً، قم ببرمجة أوامر الـ Cut، Copy، وPaste بإستدعاء clipboard procedures . على سبيل المثال، يمكنك إدخال Memo في الـ ChildWin form، وإستدع الـ CutToClipboard method للـ object . أنظر الباب السادس عشر لمزيد من المعلومات عن برمجة الـ clipboard .

* إدخل OnCloseQuery في الـ ChildWin module . حدد الـ CanClose بـ False لمنع المستخدمين من إغلاق نوافذ دون حفظ التغييرات .

افكار للمستخدم الخبير

* نافـذة واحـدة فـقط فى تطبـيق الـ MDI يمكن أن تكـون الـ FormStyle الخاصة بها محددة بـ fsMDIForm . ويجب أن تكون هذه النافذة الرئيسية للتطبيق (إستخدم الـ Project|Options إذا لزم الأمر لتحدد أى form هى الرئيسية) .

* إذا ظهـرت الـ Child window الخاصـة بالبـرنامج كنوافـذ مسـتقلة على شاشـة الحاسـب المكتبى ، بدلاً من أن تظهر داخل النافذة الرئيسية ، فإن خاصية الـ FormStyle لـ Child window form تكون غالباً غير محددة بشكل سليم . (فى النسخ السـابقـة من Delphi و Windows) يمكن أن تقـود هذه المشكلةإلى خطأ حـماية عـامة ، ولكن الأمر لم يعـد كـذلك) . يجب أن تكون كل Child forms window لهـا خـاصيـة الـ FormStyle مـحـددة بـ fsMDIChild . إن النافـذة الرئيسـيـة للبـرنامج يجب أن يكون الـ FormStyle الخـاصـة بهـا مـحـددة بـ fsMDIForm ، ويجب أن تكون نافذة واحدة لكل تطبيق تملك هذه الصفة .

* إذا لم تكن تريد أن تظهر الـ Child window بصورة تلقائية عندما تقوم بتشغيل التطبيق ، أختر Options|Project وأنقل الـ Child window إلى قائمة الـ Available forms . ولكن ، لا يوجد خطأ من الناحية الفنية فى أن يقوم التطبيق بإنشاء الـ Child window الأولى تلقائياً . على سبيل المثال ، يمكن أن يمنح محرر نص الـ MDI المستخدمين نافذة خالية فى البداية .

يستطيع التطبيق إنشاء الـ Child window إضافية كما هو موضح فى هذا الباب - يتم إنشاء النافذة الأولى فقط بصورة تلقائية عندما يبدأ البرنامج .

* إذا كان إغلاق الـ Child window لتطبيق الـ MDI يجعلها تصغر ، تأكد من أن module الـ Child window لها OnClose الذى يحدد الـ Action بـ caFree . إذا لم تقم بتعيين هذه القيمة للـ Action ، لا يقوم التطبيق بتحرير object أخرى فى وقت التشغيل (مثل TChildForm فى هذا الباب) ، تذكر تعيين حالات الـ object التى تم إنشاؤها فى المتغير الحالى - لا تستدع الـ Create method بإستخدام المتغير الحالى الذى لم يتم بدؤه . على سبيل المثال ، هذا الخطأ كثيراً ما يحدث :



```
var
    Child: TChildForm;
begin
  Child.Create(Application);  {.???}
  ...
```

والعبارة الخاطئة تحاول أن تجعل الـ Child window تولد نفسها ، والذى لا يمكن أن يحدث . وغالباً ما تؤدى العبارة إلى خطأ "access violation" أو إنتهاك الوصول لأن الـ Child لم يبدأ وإستدعاء object methods لم يتم بدئه ، يعد ذلك خطأ برمجة . إستخدم الـ code التالية لتستدعى الـ Create method للـ class وتعين الـ object الناتج إلى الـ Child reference :

```
var
  Child: TChildForm; // Or, you may use TForm as the type
begin
  Child := TChildForm.Create(Application);
```

* ليس مطلوب منك أن تحفظ object references نافذة صغيرة مع الـ Application كنافذة أم يؤدى إلى إنشاء النافذة الصغيرة . إن النافذة الأم مسئولة عن المحافظة على وتحرير الـ object الناتج . وكمثال على هذا المبدأ ، يمكنك تقصير الـ CreateChild procedure فى الـ Main module للـ MDIDemo للآتى . ولكن ، هذا يجعل من الصعب أن تؤدى عمليات على الـ Child window الحديثة مثل تعيين Caption strings :

```
procedure TMainForm.CreateChild(const Name: String);
begin
  TChildForm.Create(Application);
end;
```

* فى تطبيقات الـ MDI ذات الأنواع المتعددة من الـ Child window ، قم بتسمية الـ unit modules بـ ChildXXX.pas ، حيث أن الـ XXX هى إمتداد اسم ملف الوثيقة . على سبيل المثال ، إستخدم الـ Childtxt.pas للملفات الـ txt . أو Childbmp.pas للملفات الـ bmp .



بالطبع يمكنك أن تسمى الـ modules كما تشاء ولكن هذا تقليد مفيد لتعريف الـ modules بأنواع ملفاتها .

* يكون دمج القوائم فى تطبيقات الـ MDI تلقائياً . حدد كل خصائص MainMenu AutoMerge بـ False . يجب أن تكون خاصية الـ AutoMerge لنافذة الـ MDI الأم محددة بـ False . إستخدم خاصية الـ AutoMerge للقيام بدمج القوائم فقط فى غير تطبيقات الـ MDI .

* تجنب إسقاط components على الـ form الرئيسية للـ MDI . إن الـ controls خفيفة الوزن مثل الـ TLabel لن ترسم من الأساس ، و handle controls النافذة العادية (مثل الـ TButton) قد لا تعمل بشكل صحيح عند إسقاطها على client area للـ MDI form الرئيسية . وهذا بسبب التلاعب فى واجهة التطبيق للـ Windows MDI . ولكن ، الـ TPanels العلوية أو السفلية المعنية تعتبر مقبولة . أضف control objects أخرى على الـ Child window أو فى الـ Panel objects .

المشروعات التى يمكنك تجربتها

١٥-١ : أضف Child window metafile للـ MDIDemo .

١٥-٢ : أكتب تطبيق تصفح ملف يعمل مع مجموعة متنوعة من الملفات .

١٥-٣ : قم بتصميم أمر File|Save All ظاهرى فى Child class window .

١٥-٤ : متقدم . أضف أمر Window|Restore لتطبيق الـ MDIDemo . يجب أن يقوم الأمر بإستعادة Child window إلى مواضعها وأحجامها الأولى عقب أمر Tile أو Cascade أو Minimize all تذكر إستخدام البرمجة المختصة بالـ object يجب أن تكون Child window objects قادرة على إستعادة نفسها إلى موضع وحجم محفوظ . لا تكلف الأم بهذا العبء؛ إجعل الصغار يعملون .

١٥-٥ : أكـتـب مـحـرر نص MDI بإسـتـخـدام MDI Application template . لن تكون النتائج خارقة ، ولكن التجربة ستجعلك تفهم تقاليد التسمية المتنوعة للـ template .

١٥-٦ : متقدم . قم بتحويل الـ MDIDemo إلى MDI Application template مـحـسـن إدخل الـ virtual methods مـثـل الـ LoadData والـ SaveData فى الـ TChildForm class لا تقم بإنشاء object من تلك الـ class ، بدلاً من ذلك ، قم بتقسيم فرعى لـ forms جديدة على أساس الـ TChildForm وأكتب الـ SaveData procedures وLoadData لقـراءة وكـتـابة أنواع معينة من الملفات . لكى تبرمج تطبيقاً جديداً ، فإنك تضيف form Child window جـديدة ، أكـتـب LoadData proceducers وSaveData وقم بإنشاء نماذج صغيرة فى الـ Main module .

ملخص :

* إن تطبيقات الـ MDI تستخدم الـ TForm الخاص بـ Delphi ، وهو غير موجود على لوحة الـ VCL palette ، لإنشاء نوافذ رئيسية وChild window . وتحل النافذة الرئيسية محل نوافذ الـ client والإطار المستخدمة فى برمجة الـ Windows الكلاسيكية .

* إن كل تطبيق MDI له ثلاثة أجزاء أساسية : form نافذة رئيسية ، form واحدة أو أكثر من Child window الوثائقية ، وقائمة رئيسية .

إستخدم خاصية الـ WindowMenu للـ form الرئيسية لتحدد بند قائمة رفيعة المستوى لعناوين نافذة القائمة .

* حدد خاصية الـ FormStyle للـ form الرئيسية بـ fsMDIForm . يمكن أن يكون هناك form واحدة فقط من هذا النوع فى كل تطبيق . فى كل Child window ، حدد الـ FormStyle بـ fsMDIChild يمكنك إنشاء أى عدد من أنواع Child window كما يحتاج تطبيقك .



* لإنشـاء Child window فى وقت التشـغيل ، قم بإسـتدعـاء الـ Create method الخـاص بـ class النافـذة قم بتـمـرير الـ Application إلى الـ object parameter الأم للـ Create method .

* يجـب أن تقـــوم كل Child window بتـنفـــيـــذ OnClose وOnCloseQuery إستخدم هذه الـ events لتحذر المستخدم حول البيانات التى لم يتم حفظها عندما يحاول إغلاق النوافذ .

* إن خاصية الـ ActiveMDIChild الخاصة بالـ form الرئيسية تشير إلى Child window الحـاليـة . إذا كـانت هذه الخـاصيـة nil ، فـلا يوجـد Child window . قم دائماً بفحص ما إذا كانت الـ ActiveMDIChild مساوية لـ nil قبل إستخدام هذه الخاصية لإستعاد method وتعيين قيم خاصية .

* إن خاصية الـ MDIChildCount التابعة للـ form الرئيسية تساوى عدد النوافذ الصغيرة . إذا كانت الـ MDIChildCount مساوية لصفر ، فلا يوجد Child window .

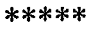
* إن الـ MDIChildren array يوفـر وصـولاً مـفـهـرسـاً لكل الـ Child window المملوكة للتطبيق .

* إن الـ MainMenu للـ Child window تندمج بصـورة تلقـائيـة مع الـ MainMenu الخاص بالـ form الرئيـسيـة . إسـتـخـدم خـاصيـة الـ GroupIndex للتحكم فى الدمج .

* يشير الـ Handle الخاص بالـ form الرئيسية إلى نافذة الإطار لتطبيق الـ MDI . تشير خـاصيـة الـ ClientHandle إلى نافـذة العـميل . لا تحتـاج أغلب تطبيقات الـ MDI إلى إستخدام هذه الخصائص .

* إستخدم template مشروع الـ MDI Application حتى تنشئ بسرعة مشروعات الـ MDI ذات قوائمية إفتراضية ، toolbar ، وstatusbar .

إن نقل البيانات من تطبيق إلى آخر يعتبر أمراً صعباً فى برمجة Windows . إن الباب التالى يساعدك على خوض ما لم تقنيات نقل بيانات الـ clipboard ، والـ DDE ، والـ OLE . وهو أيضاً يوضح إستخدام وإنشاء الـ ActiveX controls .





الباب السادس عشر
التطوير مع الـClipboard، الـDDE، والـOLE

محتويات هذا الباب:

- Components.
- نقل بيانات الـClipboard.
- الـTClipboard class.
- Dynamic Data Exchange (DDE).
- Object linking and embedding (OLE).

إن تشارك البيانات بين التطبيقات أمراً ليس سهلاً. يمكنك أن تقود code إلى بيئة شبكية أو متعددة المهام، ولكن أن تجعلها تتشارك فى البيانات مع برامج أخرى، فهذا يأخذ مزيداً من العمل والمراوغة.

يوفر الـ Windows ثلاثة methods اساسية للتشارك فى البيانات: clipboard، الـ Dynamic Data Exchange (DDE)، والـ Object Linking and Embedding (OLE). وأى method تستخدمها فهذا يعتمد على احتياجات تطبيقك ونوع البيانات التى يجب عليك ان تتشارك فيها. فى هذا الباب، سوف اوضح كيفية استخدام الـ clipboard لنقل النص والجرافيك، وكيفية استخدام الـ DDE لإقامة روابط مع برامج مثل الـ Microsoft Excel و Word، وكيفية ربط وتضمين الـ OLE فى تطبيقات Delphi.

:Components

فـيـمـا يلى قـائمـة بـ Delphi Components للتطوير مع clipboard والـ DDE والـ OLE:

• DDe ClientCouv- إن تطبيق عـمـيل DDE (وهو تطبيق يتلقى بيانات من مساعد الـ DDE) يستخدم هذا الـ Component لإقامة رابطة، تسمى محادثة [أو conversation]، مع server. كل تطبـيق client له علـى الأقل واحـداً من هذه الـ PaletteSystem ibject .

• DDe Client Item: بالإضـافـة إلى الـ Dde Client Couv object، يسـتـخـدم تطبـيق الـ DDE client واحـداً أو اكـثـر من الـ DDe Client Item كحاويات تتلقى البيانات من الـ server. كل تطبيق client له على الأقل واحداً من هذه الـ objects لكل DdeClientConv . Palette: System

• DdeServerConv- إن تطبيق DDE (server) (تطبيق يرسل بيانات لـ DDE client) يمكن أن يستـخـدم هذا الـ component لخلق مـحـادثة DDE. إن اغلب تطبيقات الـ server تستخدم على الأقل واحداً من هذه الـ objects، ولكن من الممكن خلق محادثة بدون الـ DdeServerConv . Palette: System .

• DdeServerItem- يستخدم تطبيق أو Server الـ DDE واحداً أو أكثر من هذه الـ objects لإرسال بيانات للـ client. إن الـ DdeServerItem يمكن ربطه أو عدم ربطه بالـ DdeServerConv، ولكنه غالباً ما يكون مرتبطاً به. Palette: System

• OleContainer- استـخـدم هذا الـ object لإنشاء حـاوية OLE يمكن ان تحـمل objects من الـ OLE servers. علـى سـبـيل المثـال، باسـتـخـدام الـ OleContainer، يمكن لوثائق تطبيـقك ان تجـمع لوحـات الـ Excel، ووثائق الـ Word، ورسوم الـ Visio Express. يستطيع المستخدمون يعرض بيانات فى هذه الـ objects مباشرة من تطبيقك. Palette: System.

انتقالات البيانات عبر الـ Clipboard:

كما يعرف اغلب مستخدمى الـ Windows، تقوم اوامر الـ Cut، Copy، Paste الخاصة بقائمة الـ Edit بنقل البيانات من وإلى الـ Windows clipboard .



وغالباً ما يكون الاستخدام الشائع لهذه الأوامر هو نقل النص بين التطبيقات، أو حتى داخل نفس التطبيق، ولكن Clipboard يمكن أن تتولى امر نقل البيانات فى Forms اخرى لا حصر لها. على سبيل المثال، يستطيع المستخدمون قص ولصق الـ bitmaps بين برامج الجرافيك.

إن الـ Edit والـ MaskEdit والـ Memo التابعة لـ Delphi (وكذلك الـ TDBEdit components، TDBImage، وTDBMemo) تدعم نقل البيانات من خلال الـ clipboard لنص احادى العبارة. لنقل بيانات الـ object، استدع methods مثل الـ CutToClipboard، CopyToClipboard، PasteToClipboard، إستجابة لإختيار المستخدم لأمر قائمة أو ضغط زر. على سبيل المثال، لقص نص مختار من الـ Memo، استخدم العبارة:

```
Memo1.CutToClipboard;
```

والـ two procedures الآخران ايضاً يسهل استخدامهما:

```
Memo1.CopyToClipboard;
Memo1.PasteFromClipboard;
```

Tip

فكرة: للـ components المؤسسة على نص، فإن النص المختار فقط يتم نسخة أو قصة إلى الـ clipboard. لنسخ أو قص كل text object، استدع Memo1.SelectAll قبل أداء عملية الـ clipboard.

الـ TClipboard Class:

لقص ولصق بيانات غير نصية، استخدم الـ TClipboard Class فى الـ Clipbrd unit (لاحظ الهجاء المختصر). لأن الـ TClipboard ليست component على الـ VCL palette، فلكى تستخدم هذا الـ class يجب ان تضيف اسم الـ unit لتعريف الـ uses للـ module. قم بتعديل التعريف كما يلى:

```
Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs, Clipbrd;
```

خصائص الـ TClipboard:

توفر الـ TClipboard الخصائص التالية:



• AsText– استخدم خاصية الـ string هذه لنسخ ولصق النص من وإلى الـ clipboard ، قم بتـعـيين Pascal string للـ AsText . ولكى تتلقى الـ string ، استخدم الـ string فى عبارة . لقراءة وكتابة نص clipboard اطول ، يمكنك ايضاً ان تسـتـخـدم الـ SetTextBuf methods والـ GetTextBuf للـ TClipboard ولكن لأن الـ Object Pascal strings يمكن أن تكون الآن غير محددة فى الطول ، فهذه الـ methods لم تعد على نفس درجة الإفادة كما كانت فيما مضى . إذا كانت الـ clipboard تحتوى على نص– فمن الحكمة دائماً ان تفحص هذا قبل اللصق– فتكون عبارة الـ If التالية صادقة :

```
if Clipboard.HasFormat(cf_Text) then ...
```

• FormatCount– إن خاصية العدد الصحيح هذه تساوى عدد العناصر فى الـ Formats array، الموضح بعد ذلك . إن تعبيرات الـ array الصالحة تتراوح فيما بين الـ Formats[0] إلى Formats[FormatCount - 1] . اســتـخـدم دائمـاً الـ FormatCount لمنع اخطاء مدى الفهرس عند استخدام الـ Formats array .

• Formats– هذا هو Array من قـيم الـ Word التى تمثل Forms أو أنواع البيانات المسجلة الخاصة بالـ clipboard . إن خاصية الـ FormatCount تساوى عدد العناصر فى الـ Formats array . يذكر الجدول ١٦-١ ثوابت الـ clipboard format المعيارية من الـ Windows.pas المتوفر فى الـ source files لـ Delphi . توجد الثوابت فى الحروف الكبيرة بالملف ، ولكن يمكنك أن تكتبـهـا بالحروف الصغيرة إذا أردت .

تنسيقات إضافية:

تسـجل الـ Clipbrd unit اثنين من تنسيقات الـ clipboard بالإضـافـة إلى التنسيقات المعيارية المذكورة فى جدول ١٦-١ . تقوم الـ unit بهذا عن طريق تنفيذ هذه العبارات فى أى برنامج يستخدم الـ Clipbrd :

```
cf_Picture := RegisterClipboardFormat('Delphi Picture');
cf_Component := RegisterClipboardFormat('Delphi
Component');
```



جدول ١٦-١ تنسيقات الـ Clipboard من الـ WinTypes.pas.

Constant	*Format Value*
CF_TEXT	1
CF_BITMP	2
CF_METAFILEPICT	3
CF_SYLK	4
CF_DIF	5
CF_TIFF	6
CF_OEMTEXT	7
CF_DIB	8
CF_PALETTE	9
CF_PENDATA	10
CF_RIFF	11
CF_WAVE	12
CF_UNICODETEXT	13
CF_ENHMETAFILE	14
CF_HDROP	15

استخدم متغيرات الـ cf_Picture والـ cf_Component، مع الـ HasFormat الخاصة بالـ clipboard، لتحدد إذا ما كانت الـ clipboard تحتوى الآن على صورة TGraphics (أيقونة، bitmap، أو ملف حجم كبير) أو نوع آخر من الـ component object . إن Delphi نفسه يستخدم cf_Component لقص ولصق الـ component objects على الـ forms، ولكن التطبيقات لا تحتاج إلى استخدام هذا المتغير.

وهذا هو مثال لـ custom clipboard format، والذى يمكنك ان تفعله فى تطبيقاتك. على سبيل المثال، استخدم هذه التقنيات لتسجيل structure format

مـعين لنقل بيـانات من هذا الـنوع إلى تطبيقـاتك اخـرى باسـتخـدام الـ clipboard . object

الـ TClipboard methods:

استـدع الـ TClipboard class لقص ونسخ ولصق بيانات، وكـذلك لأداء عـمليـات clipboard متنوعـة. توفـر الـ Clipbrd unit الـ Clipboard object جـاهز للاسـتخـدام- لا يجب عليك إنشـاء object من الـ TClipboard class. أضف لأمر الـ Uses بالـ module. يمكنك ان تفعل هذا بواسطة تعديل تعـريف الـ Uses للـ module. على سـبـيل المثـال، اضف النص التـالى (الموضح بالخط السميك):

```
implementation
uses Clipbrd
```



تحذير: انتبه إلى كيفية هجاء الـ Clipbrd. لا تفعل كما فعلت أنا - كتبت **uses Clipboard**، ثم أضعت عشر دقائق اتعجب لما لم يقم برنامجى بالـ compile.

إن استخدام الـ Clipbrd unit بهذه الطريقة يجعل أى تعريف عام متاح للـ procedures والـ functions الموجـــــــودة فى قطاع الـ implementation module. على سبيل المثـال، فى procedure أو function، يمكن لعبـارة أن تستدعى الـ HasFormat method الخاص بالـ Clipbrd unit مثل هذه:

```
if Clipboard.HasFormat(cf_Bitmap) then
    // ... copy bitmap from clipboard
```

لاحظ ان الاستدعاء للـ HasFormat هو فى اشارة للـ Clipboard، والذى قد تم بدءه بالفعل وجاهز للاستخدام. إذا لم تكن تريد أن تكتب Clipboard مرات ومرات، يمكنك استخدام عبارة with كما فى هذه الجزئية:

```
with Clipboard do
begin
  if HasFormat(cf_Bitmap) then
      // ... copy bitmap from clipboard
      else if HasFormat(cf_Text) then
```

```
    // ... copy text from clipboard
  end;
```

فيما يلى وصف مختصر للـ TClipboard methods الرئيسية التى يمكنك استدعائها بالاشارة للـ Clipboard :

- Assign-To : لنسخ الجرافيك إلى Clipboard، قم بالتمرير للـ Assign أى TGraphic، أو TBitmap، أو TPicture، أو TMetafile أو أى خاصية object.
- Clear : يسمح لأى بيانات فى الـ clipboard . لا يجب عليك عادة ان تستدعى هذا الـ method، لأن تعيين معلومات جديدة إلى الـ clipboard . يقوم بأداء مسح Clear . ولكن، قد تستدعى الـ Clear قبل إنتهاء البرنامج مباشرة إذا كانت الـ clipboard بها مجموعة كبيرة من البيانات. إسئل المستخدم عما إذا كانت الـ clipboard يجب مسحها.
- Close : يغلق الـ clipboard عقب استدعاء الـ Open. كل عبارة Open يجب أن يكون لها عبارة Close مقابلة لها. لا يجب عليك استدعاء الـ Open والـ Close إلاّ إذا وصلت للـ clipboard باستدعاء Windows API. إن الـ Windows API فى الـ Clipboard تفتح وتغلق الـ Clipboard بصورة تلقائية على حسب الحاجة.
- GetAsHandl : يُعاد الـ handle إلى بيانات الـ clipboard. استخدم هذا فقط إذا احتجت إلى استدعاء الـ Windows API التى تتطلب handles، أو عند تحويل البرامج التى كتبت بطريقة تقليدية إلى Delphi. معظم التطبيقات لا تحتاج إلى استخدام هذه الـ function.
- GetComponent : تستعيد الـ Delphi component من clipboard. لمزيد من المعلومات حول استخدام الـ GetComponent والـ SetComponent انظر الـ Component objects والـ clipboard، فى هذا الباب.
- GetTextBuf : هذه الـ function تستعيد النص من الـ clipboard كـ null-terminated buffer. فى الماضى، كانت هذه الـ method تستخدم مع

الكميـات الكبيـرة من النص ، لأن الـ Pascal strings ليس لهـا حـد فى الطول ، فيمكنك ببساطة ان تستخدم خاصية الـ AsText لاستعادة نص الـ clipboard . " انظر النص و الـ clipboard" فى هذا الباب . لمزيد من المعلومات حول استخدام الـ GetTextBuf و الـ SetTextBuf .

• HasFormat : ترجع بـ True إذا كـانـت الـ clipboard لهـا بيـانات بـ format عُرفت من خلال Word argument دائماً أستدعى HasFormat قبل نسخ معلومات من الـ clipboard فى تطبيق أو متغير آخر .

• Open : انك تحتـاج لفتح الـ clipboard باستـدعـاء هذا الـ Windows فقط إذا استـدعـيت الـ Windows API للوصـول الى الـ clipboard . إن كل الـ methods فى الـ clipboard تفتح clipboard بصورة تلقائية على حسب الحاجة .

تحذير : إن استدعاء الـ Open يمنع بشكل مؤقت المهام الأخرى من استخدام الـ clipboard . وكل استـدعـاء للـ Open يجب أن يكون له استـدعـاء مـقـابل للـ Close .

• SetAsHandle : يُعين بيـانات للـ clipboard باستـخـدام Windows handle . انظر الـ GetAsHandle . ان غالبية التطبيقات لاتحتاج إلى استدعاء هذا الـ method .

• SetComponent : يـعـين component إلـى الـ clipboard . انـظـر الـ GetComponent و Component clipboard فى هـذا البـاب لمعـرفـة المزيد من المعلومات .

• SetTextBuf : يعين بيانات null-terminated للـ clipboard . استخدم هذه الـ function للعمل مع كميات كبيرة من النص المخزن فى الـ Char arrays . انظر أيضـاً الـ GetTextBuf . لأن الـ Pascal strings اصبـحـت الآن فـعليـاً غـيـر محددة فى الطول ، فإن اغلب التطبيقات يمكنها ان تعين string لخاصية AsText التابعة للـ Clipboard بدلاً من استدعاء الـ SetTextBuf و الـ GetTextBuf .

النص و الـ clipboard :

من السـهل نقل النص من والى الـ clipboard والـ components على نص مثل الـ Memo والـ Edit باستخدام نوع بيانات الـ string الديناميكى للـ Pascal ،

فبامكانك ان تعين أى خاصية Text لخاصية الـ AsText التابعة للـ Clipboard . ولكن هناك بعض النقاط التى تستحق مزيداً من الاعتبار .

لعرض هذه الجوانب والـ clipboard فان الخطوات التالية تستخدم الـ Memo . بالرغم من أنه يمكنك بسهولة أكثر ان تؤدى عمليات مشابهة باستدعاء CopyToClipboard ، CutToClipboard ، PasteFromClipboard التابعة للـ Memo، فالخطوات التالية تقدم تقنيات برمجة هامة يمكنك تطبيقها على أنواع أخرى من عمليات نقل بيانات الـ clipboard . اتبع هذه الخطوات :

١- ابدأ تطبيقاً جديداً. اضف تعريف uses للـ Clipbrd بعد كلمة الـ implementation الأساسية فى الـ form unit. للتطبيق . يجب ان يبدء الـ تطبيق هكذا :

```
implementation
uses Clipbrd;
```

٢- أضف Memo من الـ Standard palette على الـ form . اختر Memo1 وحدد خاصية الـ ScrollBars له بـ ssBoth . يمكنك تكبير الـ Memo1 إذا أردت .

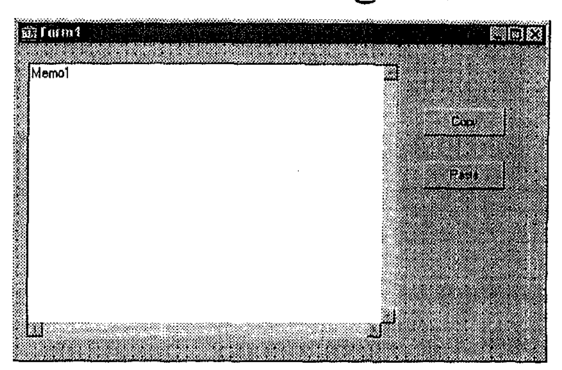
٣- أضف اثنين من الـ Button على form . قم بتغيير خاصية الـ Name للزر الأول ليصبح CopyButton والـ Caption الخاص به ليصبح Copy . قم بتغيير خاصية الـ Name للزر الثانى ليكون PasteButton والـ Caption الخاص به ليكون Paste . شكل (١٦-١) يوضح نافذة البرنامج حتى الآن .

٤- اضغط مرتين زر الـ Copy لإنشاء الـ OnClick الخاص به . إدخل الـ code من القائمة (١٦-١) فى الـ procedure على القرص المدمج ، يوجد هذا النص فى ملف الـ Clip1.pas فى دليل الـ Source\Clipboard .

٥- اضغط مرتين زر الـ Paste لإنشاء الـ OnClick الخاص به . ادخل الـ code من القائمة (١٦-٢) فى الـ procedure على القرص المدمج ، يوجد هذا النص فى ملف الـ Clip1.pas فى دليل الـ Source\Clipboard .

٦- قم بتشغيل البرنامج الاختبارى واستخدم الـ Windows Notepad utility لفتح ملف نص . اختر كل السطور ، واختر الـ Edit|Copy لنقل النص إلى

clipboard. ارجع مرة أخرى إلى البرنامج الاختبارى، واضغط زر الـ Paste لنسخ نص الـ clipboard إلى الـ Memo. اضغط Copy لنسخ النص إلى الـ clipboard . احذف النص، واضغط Paste لتتحقق من ان البرنامج يلصق الـ clipboard إلى الـ Memo . يمكنك ايضاً أن ترجع للـ Notepad، وتختار الـ Edit|Paste لإدخال النص المنسوخ من الـ clipboard .



شكل (١٦-١) الـ Form window الخاصة باختبار الـ clipboard مع الـ Memo والـ Buttons.

القائمة ١٦-١: OnClick لزر الـ Copy.

```
procedure TForm1.CopyButtonClick(Sender: TObject);
begin
 with Memo1 do
   begin
    if SelLength = 0 then   // If no text selected,
       SelectAll;          // select all lines of Memo1.
         if SelLength = 0 then   // If still no text selected,
     Exit;                // Memo1 is empty-exit now.
      Clipboard.Clear;
    Clipboard.AsText := Memo1.Text;
   end;
end;
```

القائمة ١٦-٢: OnClick لزر الـ Paste.

```
procedure TForm1.PasteButtonClick(Sender: TObject);
  begin
   if Clipboard.HasFormat(CF_TEXT) then
    Memo1.Text := Clipboard.AsText;
end;
```

تقنية نص clipboard:

هناك تقنية قديمة للعمل بنص clipboard والتى لا تزال متاحة. يمكنك استخدام هذا الـ method إذا كنت تكتب code للـ Windows 3.1، أو إذا كان لديك نسخة أولية من Delphi. ويمكنك ايضاً استخدام المعلومات الواردة فى هذا الفصل لتحديث البرامج القديمة التى تستخدم هذه التقنية حالياً لتخزين نص الـ clipboard فى الـ character arrays.

استدع الـ SetTextBuf method الخاص بالـ Clipboard لتنسخ Buffer النص الكبير إلى الـ Clipboard. على سبيل المثال، إذا كان الـ PChar Buffer يشير إلى null-terminated string buffer، فهذه العبارة تنسخ النص إلى الـ clipboard.

```
Clipboard.SetTextBuf(Buffer);
```

لاستدعاء النص من الـ clipboard، قيم بتخصيص ذاكرة للـ buffer واستدع الـ GetTextBuf function الخاصة بالـ Clipboard، والتى ترجع عدد الـ Clipboard المنسوخة.

```
Buffer := StrAlloc(1024);
Len := Clipboard.GetTextBuf(Buffer, 1024);
```

On the CD-ROM

على القرص المدمج: لسوء الحظ، إن الـ GetTextBuf يطلب منك ان تحدد مسبقاً حجم buffer النص، ولكن الـ Clipbrd unit لا تقدم أية methods لتحديد هذا الحجم. لمعرفة السبيل إلى الخروج من هذه المشكلة، اتبع الخطوات المذكورة فى الفصل السابق لإنشاء برنامج اختبارى له Memo واثنين من الـ Buttons يسميان CopyButton وPasteButton. بدلاً من

استخدام OnClick للـ Button فى القائمة السابقة، استخدم الـ code الموجودة فى القـائمـة ١٦-٣ والقـائمـة ١٦-٤ . يوجـد هذا النص على القـرص المدمج فى دليل Source\Clipboard.

القائمة ١٦-٣: OnClick هذا يعرض كيفية استخدام الـ SetTextBuf method الخاص بالـ Clipboard لنسخ null-terminated string buffer إلى الـ clipboard.

```
        { Copy null-terminated text buffer to clipboard }
procedure TForm1.CopyButtonClick(Sender: TObject);
var
  P: PChar;  { Pointer to character buffer }
begin
  with Memo1 do
     begin
      if SelLength = 0 then   // If no text selected,
           SelectAll;          // select all lines of Memo1.
             if SelLength = 0 then   // If still no text selected,
           Exit;              // Memo1 is empty-exit now.
             P := StrAlloc(SelLength + 1); // Allocate char
     buffer
         try
           GetTextBuf(P, SelLength + 1); // Copy text to buffer
              Clipboard.SetTextBuf(P); //Copy text buffer to
          clipboard
             finally
           StrDispose(P);  // Dispose of character buffer
             end;
        end;
end;
```

القائمة ١٦-٤: OnClick هذا يعرض بديل للـ GetTextBuf method بالـ Clipboard، والذى يتطلب منك ان تحدد حجم الـ buffer مسبقاً.

```
{ Copy null-terminated text buffer from clipboard }
procedure TForm1.PasteButtonClick(Sender: TObject);
var
  Data: THandle;
```

```
    DataPtr: PChar;
     P: PChar;
  begin
     Clipboard.Open;
     try
      Data := GetClipboardData(cf_Text);
        if Data = 0 then Exit;
        DataPtr := GlobalLock(Data);
        try
         P := StrNew(DataPtr);
           Memo1.SetTextBuf(P);
          finally
         GlobalUnlock(Data);
           StrDispose(P);
          end;
       finally
      Clipboard.Close;
       end;
  end;
```

تتأكد القائمة ١٦-٣ من أن الـ Memo1 له بعض النص المختار- إذا لم يكن، يقوم الـ procedure باستدعاء الـ SelectAll . هذه الخطوة اختيارية، ولكنها تجعل من السهل ان تنسخ كل الوثيقة إلى الـ clipboard دون ان تطلب من المستخدمين ان يختاروا كل رمز . يقوم الـ StrAlloc باعداد الـ buffer الذى ينسخ الـ GetTextBuf بداخله نص الـ Memo1 . يستدعى البرنامج بعد ذلك الـ SetTextBuf الخاصة بالـ Clipboard لنسخ ذلك الـ buffer إلى الـ Windows clipboard .

عند القيام بتخصيص أماكن فى الذاكرة، من الحكمة دائماً ان تدخل عبارات متتابعة فى الـ try-finally . كما توضح القائمة ١٦-١ ، فإن الذاكرة المخصصة للـ P تم تنظيمها بواسطة الـ StrDispose بغض النظر عما إذا كانت أى عبارة فى الـ try تثير exception .

وتدور القائمة ١٦-٤ حول الـ Clipboard lack للـ method لتحديد كم النص الذى تحمله الـ clipboard . فبدلاً من استدعاء الـ GetTextBuf لاستدعاء

النص من الـ clipboard، يقـوم الـ procedure باسـتـدعـاء الـ Windows API للحصول على هذه الـ Windows API، وهى الـ GetClipboardData.

إن الـ try الداخلى للـ procedure يستدعى الـ StrNew لتخصيص ذاكرة للـ P الـ PChar pointer، ثم ينسخ بيـانات الـ clipboard كـ null-terminated string إلى تلك الذاكرة. وللعرض فقط، يقوم الـ procedure بتمرير الـ string للـ Memo.

Caution

تحذير: عند استخدام نقل clipboard method فى القائمة ١٦-٤، يجب أن تسـتـدعى الـ Open والـ Close كـمـا هو مـوضـح لأن الـ procedure يمر على الـ Clipboard methods. تذكـر، التـقنيـات المذكورة فى هذا الفصل توضح methods - لا تستخدمها فى تطبيقات جديدة.

الجرافيك والـ Clipboard:

يمكنك استخدام الـ Assign method للـ Clipboard لقص، نسخ، ولصق الـ TGraphics مـثل أولئك الموجـودون فى خـاصـيـة الـ Picture للـ Image. لتـوضـيح التـقنيـة، إبدأ تطبـيـقـاً جـديداً أو أضف تعـريف Uses إلى قطاع الـ

```
implementation
uses Clipbrd;
```

اضـف اثـنين مـن الـ Image مـن Additional palette و Button مـن Standard palette على form. حدد خاصية الـ Stretch للـ Images بـ True. اضـغط مرتين الـ Button وادخل العـبارات التـاليـة بين الـ begin و الـ end فى OnClick للـ object. (سـوف تؤدى هذه العـمليـات فى امـاكن منفـصلة فى البـرامج أو فى تطبيـقـات منفـصلة. ولكن للعرض، قم بكلا الانتـقـالين فى وقت واحد).

```
Clipboard.Assign(Image1.Picture);  // Copy to clipboard
if Clipboard.HasFormat(cf_Picture) then
  Image2.Picture.Assign(Clipboard); // Copy from clipboard
```

مازلنا فى Delphi، اختر Image1 واستخدم خاصية الـ Picture التابعة له لتحميل ايقونة كـ Bitmap، أو ملف metafile. قم بتشغيل البرنامج بضغط F9،

ثم اضـغط الزر لنقل الصـورة من Image1 الى Image2 عـبـر الـ Clipboard الـ Windows . يجب ان يعرض كلا من الـ Images نفس الجرافيك .

تستدعى العبارة الأولى الـ Clipboard Assign method لنسخ خاصية TGraphics (Image1.Picture) الى الـ Clipboard تتحقق العبارة الثانية من ان الـ Clipboard لهـا cf_Picture ، وإذا كـان كـذلك ، تسـتـدعى الـ Picture.Assign method الخاص بالـ Image2 ، مما يمرر الـ argument . بالرغم من انك قـد تسـتـخـدم الـ Clipboard على انهـا للـ Assign argument بهـذه الطريقـة ، فـيـجب دومـاً ان تـخـتـبـر كـمـا هو مـوضح هنا وذلك باسـتـدعـاء الـ HasFormation مـا اذا كـانـت بيـانات الـ Clipboard مـوجـودة فى الـ form المتوقعة . على سبيل المثال ، لتلقى Bitmap منسوخة إلى الـClipboard بواسطة تطبيق آخر مثل Windows Paint utility ، استخدم Code مثل هذا :

```
if Clipboard.HasFormat(cf_Bitmap) then
  Image2.Picture.Bitmap.Assign(Clipboard)
else
    Image2.Picture.Bitmap.Assign(nil);
```

ان اسـتـخـدام خـاصـيـة الـ Picture يمكن الـ object من العـمـل مع بيـانات الأيقونة ، و Bitmap والـ metafile . وهذا أيضاً يمكن الـ object من العمل مع أى TGraphic class للاضـافـة مـثـل الـ TJPEGImage من الـ JPEGunit . إن استدعاء الـ Assign لـ TBitmap يعد أكثر تحديداً ، ولكن أى من التقنيتين تعتبر مقبولة .

Component objects و clipboard:

اسـتـدع الـ SetComponent method الخاص بالـ Clipboard لنسخ أى component الى Clipboard . اسـتـدع GetComponent method Clipboard ، أما فى نفس التطبيق أو فى تطبيق آخر ، للصق GetComponent بصرى من الـ Clipboard .

داخليـاً ، تسـتـخـدم الـ SetComponent والـ GetComponent الـ memory-streaming لنسخ component من والى الذاكـرة . لاسـتـعـادة

component، بالاضـافة إلى اسـتدعـاء الـ GetComponent يجب ان تسـجل الـ component class مع نظام الـ memory stream الخـاص بـ Delphi . والمثـال على انتقالات component clipboard . يوضح التقنيات الأساسية . اتبع هذه الخطوات لانشاء تطبيقاً اختبارياً :

١- ابدأ تطبيـقـاً جـديداً وأضف تعـريف الـ uses إلى قطاع الـ implementation كما يلي :

```
implementation
uses Clipbrd;
```

٢- من Delphi Standard الخـاصـة بـ Delphi، أضف Button و ScrollBar (الـ object الذي سيتم نسخه) على form . قم بانشاء OnClick بالضغط مرتين على الـ Button1، وادخل العبارة التالية بين الـ begin والـ end لنسخ الـ ScrollBar إلى clipboard عندما تضغط الـ Button .

```
Clipboard.SetComponent(ScrollBar1);
```

٣- قم بتشغيل التطبيق بضغط F9، ثم اضغط الزر ينسخ الـ ScrollBar1 إلى clipboard . اخرج من البرنامج لترجع إلى الـ Delphi . احفظ التطبيق فى دليل مؤقت (اختر الـ File|Save All واستخدم اسماء الملفات الافتراضية) .

٤- ابدأ تطبيقاً جديداً، وأضف Button على form، وكما سبق، أضف تعريف uses لقطاع الـ implementation .

```
implementation
uses Clipbrd;
```

٥- اضغط مرتين الـ Button1 لإنشاء OnClick ، وأدخل العبارة التالية بين الـ begin والـ end .

```
if Clipboard.HasFormat(cf_Component) then
  Clipboard.GetComponent(Self, Self);
```

٦- قم بتحريك الـ code editor إلى الـ end النهائية فى الـ unit module ، وفوق هذا السطر مباشرة، إدخل عبارة البدء التالية . يجب أن تنتهى الـ unit (متضمنة الـ end النهائية والنقطة) كما هو موضح هنا :

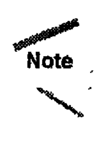


```
initialization
    RegisterClasses([TScrollBar]);
end.
```

٧- اضغط F9 لتشغيل التطبيق الثانى، واضغط الزر لنسخ الـ ScrollBar من الـ clipboard ولصقة على الـ form. (إذا لم تنجح هذه الطريقة، فقد تكون قمت بنسخ شئ آخر إلى الـ clipboard اثناء كتابة التطبيق الثانى. فى هذه الحالة، إدخل التطبيق الأول المضغوط باستخدام الـ Windows Explorer، ثم من Delphiو قم بتشغيل التطبيق الثانى واضغط الزر).

إن وظيفة الـ GetComponent الخاصة بالـ Clipboard تتطلب إثنين من الـ arguments التى تمثل الـ Parent والمالك لـ pasted object بالرغم من هذه الـ arguments غالباً ما تكون واحدة كتلك الموجودة فى المثال السابق، فإنها تختلف عندما تلصق components فى حاوية مثل الـ GroupBox. فى هذه الحالة، تكون عـادة هى المالك والـ GroupBox هو parent للـ pasted object، ويمكنك استخدام codeمثل :

```
if Clipboard.HasFormat(cf_Component) then
  Clipboard.GetComponent(Self, GroupBox1);
```



ملحوظة: فى الـ code السابق، أفترض أن العبارات موجودة فى الـ form event handler أو لـ object فى تلك الـ form، فى هذه الحالة، يشير الـ Self إلى الـ form object.

وتجدر الإشارة إلى أن الأمثلة السابقة تقدم تقنية برمجة Object Pascal مفيدة ولكن نادراً ما تحتاج إليها. عند بدء التنفيذ لأى unit module عندما يبدأ التطبيق، إدخل عبارات بين كلمة الـ initializations الاساسية والـ end النهائية للـ unit. هذا يحل محل قطاع الـ begin-end initialization ذى الاسلوب القديم فى الـ unit، والذى لايزال Delphi يوفره. فى الـ units الجديدة، استخدم الـ form الجديدة كما هو موضح هنا :

```
unit AnyUnit;
...8
initialization  { same as begin }
```

```
    // Statements to perform when application begins.
  end.
```

قـد تسـتـخـدم الـ begin أو الـ initialization ولكن الاخيـرة تسـاعـد الـ compiler على إيجاد خطأ الإزدواج لل begin والـ end الرئيسية فى مكان آخر فى الـ module، لذا تكون الـ initialization هى المرشحة .

Dynamic Data Exchange:

إن الـ Windows clipboard تعمل بشكل جيد، ولكنها بدائية .

ولكن الـ Dynamic Data Exchange (DDE) protocols تعـمل مـثل موقع Internet Web الذى يتم الوصول إليه عبر الـ modems . لنقل البيانات، يبدأ تطبيقان، معروفان بالعـميل والـ server، محادثة التى تنساب من خلالها البيانات . بعد أن تقيم التطبيقات محادثة، فيمكن أن تحدث إنتقالات الـ DDE بصورة تلقائية، أو يطلب أحد التطبيقات بيانات من الآخر .

إن مـحادثة الـ DDE تشـمل إثنين، مـرسل البـيـانات ويسـمى الـ server application، ومستقبل البيانات ويسمى الـ client application . إذا وجدت أن هذه المصطلحات محيرة، فقط تذكر أن الـ Server يقدم البيانات للعملاء . ولكن، محادثة الـ DDE هو طريق مزدوج، وخلال المحادثة، يمكن أن تنساب البيانات فى أى من الاتجاهين، مما يشوش التميز بين العميل والـ server .

إن الـ DDE مناسبة لتحديث معلومات ضمن تطبيقات متعددة . على سبيل المثال، مع الـ DDE، يمكنك كتابة برنامج يقوم بصورة تلقائية بتحديث صورة جرافيك عندما يدخل المستخدمون بيانات ورقة حسابات . أو، يمكنك استخدام الـ DDE لإنشاء تطبيقات متعددة تتشارك فى المعلومات .

Tip

فكرة مفيدة: تستطيع الـ DDE أن تمنع تدمير البيانات من جانب المستخدمين الغير خبراء . على سبيل المثال، يمكنك كتابة تطبيق عميل لعرض النتائج لحسابات ورقة الـ spreadsheet . ولكن، لا يستطيع المستخدمون استخدام تطبيق العميل لتعديل صيغ ورقة الحسابات الأصلية .

فهم الـ DDE components الخاصة بـ Delphi:

استـــتـخـــدم الـ DDE components الأربع الخـــاصـــة بـ Delphi- DdeClientConv، DdeClientItem، DdeServerConv، DdeServerItem- لإنشاء تطبيقات عميل وserver تستطيع أن تتصل باستخدام الـ DDE. والـ Coun تمثل الـ conversation والتى تمثل الصلة المقامة بين مـهـمتىّ الـ DDE. يمكنك أن تبدأ محادثة DDE عندما تصمم التطبيق، أو يمكنك أن تربط العميل و الـ server فى وقت التشغيل.

يعمل الـ DdeClientConv مثل الذى يستقبل استدعاء من حاسوب آخر. إن شكل محادثة العميل "يرفع السماعة" ليقيم رابطة مع النظام البعيد الذى أنشأ الاستـدعـاء. إن أغلب تطبيـقـات الـ DDE تحتاج إلى DdeClientConv واحـد فقط، بالرغم من أنه بامكانك استخدام object متعددة لإقامة أكثر من مسار اتصال واحد.

ويمثل الـ DdeClientItem البيـانات التى تأتى عبر الـ modem على مسار المحادثة. ان التطبيق الذى يستقبل أنواع متعددة من البيانات قد يكون له العديد من الـ DdeClientItem المرتبطة بنفس الـ DdeClientConv. ولكن، تطبيق العميل يحتاج إلى DdeClientConv واحد فقط و DdeClientItem مرتبط واحد.

فى تطبـيـق Server بعـبـارة أخـرى، مـرسـل البـيـانات اسـتـخـدم الـ DdeClientConv لبدء محادثة. انك عادة ما تحتاج إلى DdeClientConv واحد فقط لكل تطبيق، ولكن يمكنك استخدام عدة objects لاقامة محادثات متعددة. ادخل واحـداً أو أكثـر من الـ DdeClientItem لتـمـثل البيانات التى يتم ارسـالهـا للعـملاء. ان أغلب تطبيقات الـ DdeServer تحتاج إلى DdeClientConv واحد فقط DdeClientItem واحد مرتبط.

Tip **فكرة مفيدة:** لارسال واستقبال معلومات DDE فى نفس التطبيق، أضف DdeServer و DdeClient على نافـذة form المشـروع. وهذا أيضاً مفيد فى اختبار مفاهيم الـ DDE.

اختبار مصطلحات محادثة الـ DDE :

هناك ثلاث مصطلحات تعرف محادثة الـ DDE وعناصر البيانات التى تناسب بين إثنين أو أكثر من التطبيقات . وهذه المصطلحات هى :

* Service : يعُرف الـ Server ، عادة باسم الملف التنفيذى الخاص به ناقص امتداد الـ exe. فى بعض الحالات ، رغم ذلك ، يعتبر الـ DDE Service تعريف مختلف . فى تطبيق عميل الـ Delphi ، يكون الـ Service دائماً هو اسم ملف الـ Service ناقص الـ exe.

* Topic : يعرُف unit من البيانات مثل اسم الملف ، أو Caption نافذة . لايوجد قواعد محددة عما تعينه الـ Topic . فى تطبيقات Delphi، رغم ذلك ، فان الـ Topic يكون إما Caption النافذة الرئيسية لتطبيق المساعد ، أو اسم الـ DdeServerItem فى المساعد .

* Item : يعُرف عنصراً من بيانات النص المرسلة بواسطة Server إلى تطبيق عميل . يمكن أن يكون النص أى طول ، ولكن أغلب انتقالات الـ DDE تكون صغيرة . وللقوالب الكبرى من النص ، وبالأخص تلك المنظمة سطور ، تمثل الـ DDE components الخاصة بـ Items Delphi كقوائم TStrings . string

اقامة محادثات بين العميل والـ Server :

كمثال على استخدام الـ DDE ، فإنك تنشىء فى هذا الفصل تطبيقات عميل Server . ببساطة تنقل النص بين إثنين من الـ Edit controls . إن كتابة بيانات فى نافذة الـ Edit للمساعد تؤدى إلى تحديث الـ Edit الخاص بالعميل بصورة تلقائية . ولكن ، الكتابة فى نافذة الـ Edit للعميل لاتؤدى إلى تحديث الـ Server . وهذا يعد مثالاً على محادثة الـ DDE ذات الاتجاه الواحد .



على القرص المدمج: إن قوائم البرمجة فى هذا الفصل والفصول القليلة القادمة موجودة فى الدليل الفرعى Source\Dde1 على القرص المدمج .



كتابة الـ server :

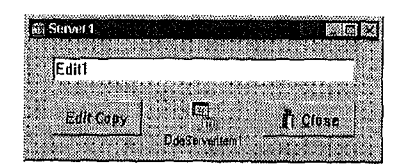
اذا لم يكن لديك تطبيق DDE server ، فمن الأفضل دائماً أن تطور server قبل كتابة العميل . كحد أدنى ، يطلب تطبيق الـ server DdeServerItem object وبعض البيانات لإرسالها للعملاء . اتبع هذه الخطوات لإنشاء مساعد بسيط يرسل نص Edit لتطبيق العميل . لقد استخدمت الاسماء الافتراضية للـ components ، والتى تساعد على توضيح العلاقات بين الـ objects والـ DDE .

١ - قم بانشاء دليل قرص جديد لعمل ملفات المشروع . ابدأ تطبيقاً جديداً وحدد خاصية الـ Name للـ Form1 بـ ServerForm . حدد خاصية الـ Caption للـ Form بـ Server1 .

٢ - اختر File|Save All واضغط المشروع فى الدليل الجديد . اجعل اسم الـ unit module بـ Server.pas والمشروع Server1.dpr .

٣ - إدخل DdeServerItem من الـ System VCL على الـ form . يرسل الـ server بيانات من خلال هذا الـ object . وكذلك ، أضف ثلاثة objects : الـ Edit ، والـ Button ، والـ BitBtn . حدد الـ Caption الخاص بالـ Button1 بـ Edit Copy . قم بتغيير خاصية الـ Kind للـ BitBtn1 لتصبح bkClose .

رتب وحدد حجم الـ form لتشبه الشكل ١٦-٢ .



شكل ١٦-٢: نافذة الـ form الـ Server1 فى Delphi.

٤ - قم بانشاء الـ OnChange الخاص بالـ Edit1 . إدخل العبارة الموجودة فى القائمة ١٦-٥ ، الـ Edit1Change procedure ، لتعيين خاصية الـ Text التابعة للـ Edit1 إلى الـ Text الخاص بالـ DdeServerItem1 . عندما يتغير نص الـ Edit1 ، فهذا يرسل النص إلى أى تطبيقات عميل تكون قد أقامت رابطة مع الـ Server .

٥- قم بانشاء الـ Button1 OnClick . إدخل العبارة من القائمة ١٦-٥ ، الـ Button1Click procedure، لإسـتـدعـاء الـ method الـ CopyToClipboard الخاص بالـ DdeServerItem . هذا يخبر تطبيقات العميل عن اسماء الـ Topic والـ Service الخاصة بالـ server . إنك تستخدم هذه المعلومة فى كتابة تطبيق عميل .

القائمة ١٦-٥: Dde1\Server.pas.

```
unit Server;

interface

uses
  Windows, Messages, Classes, Graphics, Controls,
    Forms, Dialogs, StdCtrls, DdeMan, Buttons;

type
    TServerForm = class(TForm)
      DdeServerItem1: TDdeServerItem;
        Edit1: TEdit;
        Button1: TButton;
        BitBtn1: TBitBtn;
        procedure Edit1Change(Sender: TObject);
        procedure Button1Click(Sender: TObject);
      private
      { Private declarations }
      public
      { Public declarations }
      end;

var
    ServerForm: TServerForm;

implementation

{$R *.DFM}
```

```
procedure TServerForm.Edit1Change(Sender: TObject);
begin
    DdeServerItem1.Text := Edit1.Text;
end;

procedure TServerForm.Button1Click(Sender: TObject);
begin
  DdeServerItem1.CopyToClipboard;
end;

end.
```

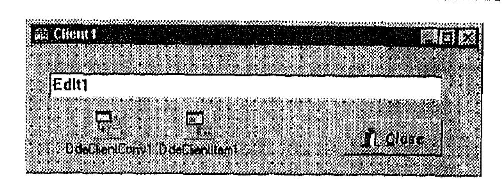
كتابة العميل:

إن كل تطبيق server يحتاج إلى واحد أو اكثر من العملاء ليشتركوا في محادثة وبث البيانات. إتبع هذه الخطوات لإنشاء تطبيق عميل للحصول على بنود نص من (server) الذى تم إنشاؤه فى الفصل السابق. فى الخطوة رقم (٢)، تأكد من أن تحفظ ملفات المشروع فى نفس دليل الـ (server).

١- إبدأ تطبيقاً جديداً. إجعل اسم الـ form بـClientForm، وغيّر الـ Caption الخاص بها ليصبح Client1.

٢- استخدم الـ File|Save لحفظ ملفات المشروع فى نفس الدليل الذى يحمل مشروع الـ Server1. إجعل إسم الـ unit module بـ Client.pas والمشروع Client1.dpr.

٣- أضف DdeClientConv من الـ System VCL على الـ form. وكذلك، أضف الـ DdeClientItem. على العكس الـ(server)، تحتاج تطبيقات العميل إلى object محادثة وعنصر لإقامة محادثة (server) ولاستقبال عناصر البيانات. لتوفير مكان لعرض البيانات التى تم استقبالها، أضف Edit معيارى على الـ form. وكذلك، أضف BitBtn وقم بتغيير خاصة الـ Kind له لتصبح bkClose. إجعل form العميل مشابهاً لشكل ١٦-٣. قد تريد تحريك الـ form إلى اسفل حتى لا تغطى بالضبط نافذة الـ (server).



شكل(١٦-٣) نافذة الـForm الـ Client1

٤- اختر الـ DdeClientItem1 (الأيقونة التى ليس بها اسهم) وحدد خاصية الـ DdeConv بـ DdeClientConv1 . هذا يقيم العلاقة بين المحادثة وشكل العنصر، ويخبر الـ DdeClientItem1 من أين يتلقى بياناته .

٥- استخدم الـ Windows Explorer لتشغيل تطبيق الـ Server . (إنك تحتاج الى أن تقوم بتشغيل كلا التطبيقين، وهو مالاتستطيع أن تفعله من داخل Delphi). اضغط زر الـ Edit Copy لتنسخ أسماء الـ Topic والـ Service الخاصة بالـ(Server) الى الـ Clipoard . يمكنك أن تترك الـ Server1 عاملاً الى نهاية هذه الخطوات .

٦- ارجع الى Delphi . اختر الـ DdeClientConv1 (الأيقونة ذات الأسهم)، واضغط الزر البيضاوى الواقع بعد خصائص الـ DdeService أو الـ DdeTopic (لايهم أى منهما). اضغط زر الـ Paste Link الخاص بالـ dialog DDE Info الناتج. يجب أن ترى الـ SERVER1 وServer1 Strings فى نافذتى التحرير . إذا لم تراهما، انتقل الى الـ Server1، اضغط Edit Copy، ثم انتقل على الفور مرة أخرى الى Delphi وحاول ثانية. إذا كنت تعرف الـ Service String والـ Topic الخاصة بالـ(Server)، يمكنك إدخالها فى الـ dialog، ولكن هذا يؤدى الى محاولة لتنفيذ (Server)و يجب أن يكون كلا الـ Strings صالحان أو يرفضهما Delphi . اضغط OK لإغلاق الـ DDE Info dialog .

٧- قم بإنشاء الـ OnCreate الخاص بالـ Form ، وادخل البرمجة من القائمة (١٦-٦) لتعيين اسم الـ DdeServerItem1 الخاص بالـ (Server) لخاصية الـ DdeItem فى الـ DdeClient Item1 الخاص بالعميل. لقد حدد العميل الآن الـ Strings (SERVER1) Service ، والـ (Server1) Topic والـ Item (DdeServerItem1) التى تتطلبها محادثة الـ DDE . يمكنك أن

تحددالـ Server والـ Topic فى وقت التصميم ، ولكنك عادة ماتعين الـ Item فى وقت التشغيل لأن ، إذا كانت خاصية الـ ConnectMode التابعة للـ DdeClientConv محددة بـ ddeAutomatic (القيم الإفتراضية) ، فإن خصائص الـ DdeServerItem1 والـ Text فى الـ DdeClientItem1 تكون عرضة للتغيير عبر الرابطة المقامة . وبسبب طبيعة الـ DD التفاعلية ، فهذه الحقيقة تكون صادقة حتى فى وقت التصميم .

٨- أخيراً ، أنشئ الـ DdeClientItem1 OnChange ، الذى يحدث عندما تأتى معلومات جديدة من الـ (Server) . ادخل العبارة من القائمة (١٦-٦) لتعيين الـ Text الخاص بالـ DdeClientItem1 (مثلاً البيانات التى تم استقبالها) الى خاصية الـ Text التابعة للـ Edit1 .

٩- أحفظ المشروع . اضغط F9 لتشغيل الـ Client1 . إذا لم يكن الـ Server1 فى حالة تشغيل بالفعل ، فإن هذا يجعل الـ Server يبدأ أيضاً (Server) . اجعل نافذتى الـ Client1 والـ Server1 مرئيتين ادخل نصاً فى نافذة الـ Edit للـ Server1 . يجب أن ترى كتابتك فى الـ ClientEdit1 . ولكن ، النص الذى يدخل فى الـ Client لاينعكس مرة أخرى الى (Server) لأن هذه هى محادثة ذات أتجاه واحد .

Note

ملحوظة : إذا واجهت صعوبات فى إقامة محادثة DDE ، إما أن تترك Delphi أو تبدأ مشروعاً جديداً وقم بتشغيل تطبيقات (Server) والعميل باستخدام الـ Windows Explorer ، أو الـ Taskbar . لأسباب غير معروفة ، يبدو أن تشغيل تطبيق عميل فى نمط الـ Debug الخاص بـ Delphi لايقيم دائماً محادثة بطريقة سليمة مع (Server) . كذلك ، حاول تشغيل (Server) قبل بدء العميل ، والذى قد يكون ضرورياً إذا كان الملف التنفيذى للـ (Server) ليس فى دليل PATH نظامى .

القائمة (١٦-٦) Dde1\Client.pas

```
unit Client;

interface
```

```
uses
 Windows, Messages, Classes, Graphics, Controls,
    Forms, Dialogs, DdeMan, StdCtrls, Buttons;

type
    TClientForm = class(TForm)
      DdeClientItem1: TDdeClientItem;
         DdeClientConv1: TDdeClientConv;
         Edit1: TEdit;
         BitBtn1: TBitBtn;
         procedure FormCreate(Sender: TObject);
         procedure DdeClientItem1Change(Sender: TObject);
       private
      { Private declarations }
       public
      { Public declarations }
       end;

var
 ClientForm: TClientForm;

implementation

{$R *.DFM}

procedure TClientForm.FormCreate(Sender: TObject);
begin
    DdeClientItem1.DdeItem := 'DdeServerItem1';
end;

procedure TClientForm.DdeClientItem1Change(Sender:
TObject);
begin
 Edit1.Text := DdeClientItem1.Text;
end;

end.
```

إقامة محادثات فى وقت التشغيل:

يوضح الفصل السابق كيفية إقامة محاثة DDE بواسطة تحديد Service وTopic Strings فى الـ Object Inspector . يمكنك أيضاً أن تبدأ محادثة DDE فى وقت التشغيل ، والذى قد يكون أكثر ملائمة ، خاصة إذا كان برنامجك يختلف مع كثير من (Servers) حول مسارات المحادثة المتعددة .

إن إقامة محادثة الـ DDE فى وقت التشغيل هى مهمة تطبيق العميل . إنك تكتب تطبيق (Server) بشكل لايختلف عما كتبته فى المثال السابق . يقوم العميل بإنشاء أسماء الـ Service والـ Topic للـ DDE ويحدد اسم عنصر البيانات الذى يتلقاه من (Server) . ولأن البرنامج يؤدى هذه الأعمال فى الـ Code ، فإنه يتعين عليه أيضاً أن يستدعى واحداً أو اثنين من الـ Methods ليربطهما بالـ (Server) .

على القرص المدمج : على سبيل المثال ، يمكنك تحويل تطبيق الـ Client1 السابق لبدء محادثة فى وقت التشغيل بحذف الـ Strings DdeService والـ DdeTopicمن الـ DdeClintConv1 . اختيارياً ، حدد الـ ConnectMode لـ ddeManual ، ثم قم بتعديل الـ event handler Form ليتوافق مع القائمة (١٦-٧) (القائمة مأخوذة من تطبيق عميل معدل فى دليل الـ Source\Dde2 على القرص المدمج) .

القائمة (١٦-٧) OnCreate المعدل يوضح كيفية إقامة محادثة DDE فى وقت التشغيل

```
procedure TClientForm.FormCreate(Sender: TObject);
begin
  if DdeClientConv1.SetLink('SERVER2', 'Server2') then
        begin
          DdeClientItem1.DdeItem := 'DdeServerItem1';
             if DdeClientConv1.ConnectMode = ddeManual then
      DdeClientConv1.OpenLink;
               end;
end;
```

OnCreate المعـدل يسـتـدعى الـDdeClientConv1 SetLink. يقـوم الـ Strings بإنشاء خصائص الـDdeDervice والـDdeTopic على التوالى- لاتقوم بتعيين String الى هذه الخصائص، استدع دائماً الـSetLink لإنشاءهم. تقوم الـ SetLink بادخال True إذا كانت قادرة على ربط الـServer المحدد. وكذلك، قم بتعيين اسم بند لخاصية الـDdeItem فى الـDdeClientItem. أخيراً، إذا كان الـ ConnectMode التابع للـDdeClientConv1 محدد بـdeManual، استدع الـ OpenLink لتـبـدأ المحـادثة. إذا كـان الـConnectMode مـحـدد بـ ddeAutomatic، فإن الـSetLink يستدعى الـOpenLink بصورة تلقائية.

تحديد الـService والـTopic :

لتـحـديد الـService والـTopic لتطبـيـق DDE server، اتبع هذه الخطوات:

١- قم بتشغيل التطبيق .

٢- اختر بعض البيانات .

٣- اختر Edit|Copy . (جرّب هذا مع الـMicrosoft Excel).

٤- انتقل الى Delphi.

٥- أضف DdeClientConv على form.

٦- اضغط الزر البيضاوى بعد خاصية الـDdeService.

٧- اضغط Paste Link فى DDE Info dialog.

هذا يجب أن يعـرض الـService string والـTopic الذى قـام (Server) بلصقها الى الـClipboard. ولكن، هذه التقنية لاتخبرك بأسماء عناصر البيانات، والتى تحتاجها لإتمام تطبيق العميل. كن مستعداً للقيام ببعض أعمال المخبرين لتـحـديد أسـمـاء بنود (Server). إن أسـمـاء البنود تحـدد بشكل نموذجى عناوين البيانات، أو المواقع الأخرى. على سبيل المثال، إن عناصر تطبيق ورقة الحسابات (Spreadsheet) تكون غالباً صفوف خلايا، عناصر قاعدة البيانات قد تكون أسماء حقل مسجل.

تلقى بيانات من الـ DDE server :

إن تطبيق عميل الـ DDE يتلقى بيانات من (server) من خلال الـ DdeClientItem. قم بتنفيذ OnChange التابع لهذا الـ Object لتستجيب للبيانات الجديدة التى تأتى من (Server). على سبيل المثال، يقوم OnChange فى القائمة (١٦-٦) بتعيين خاصية الـ Text من الـ DdeClientItem1 الى الـ Text التابعة للـ Edit.

وبالنسبة الـ blocks الطويلة من بيانات النص، خاصة عندما تنظم فى صفوف، استخدم خاصية الـ DdeClientItem Lines، وهو object من نوع الـ TStrings. على سبيل المثال، لتتلقى بند نص متعدد الأسطر فى محادثة DDE، استخدم Code مثل التالية، التى تعيّن النص الذى تم تلقيه الى خاصية الـ Lines التابعة للـ Memo:

```
with ddeClientItem1 do
  Memo1.Lines := ddeClientItem1.Lines;
```

إن أغلب تطبيقات الـ DDE تتلقى البيانات بصورة تلقائية من خلال الـ DdeClientIt OnChange. ولكن، تطبيق العميل يمكن أيضاً أن يطلب بيانات محددة من (Server) – خلية معينة فى ورقة حسابات Spreadsheet – مثلاً. لطلب بند معين من البيانات، استدع الـ DdeDlientConv RequestData function، والتى ترجع PChar pointer الى null-terminated. على سبيل المثال، هذه الجزئية من الـ Code تطلب عنصر DdeDerverItem1 وتعيّن الـ null-terminated الناتج للتعليق الخاص بالـ Label. استدع دائماً الـ StrDispose لتتخلص من الذاكرة التى تخصصها الـ RequestData للبيانات الواردة.

```
var
  P: PChar;
begin
  P := DdeClientConv.RequestData('DdeServerItem1');
  if P <> nil then
  try
    Label1.Caption := StrPas(P);
  finally
```



```
      StrDispose(P);
    end;
  end;
```

ارسال بيانات Server الـ DDE server :

إن إرسال بيانات الى عميل الـ DDE يعتبر غالباً أمراً بسيطاً من تعيين String لخصائص الـ Lines والـ Text التابعة للـ DdeServerItem . هذا يؤدى الى إرسال البيانات بصورة تلقائية الى أى عملاء يقيم الـ(Server) محادثة معهم . وبالتبادل ، يستطيع تطبيق العميل أن يعكس الأدوار ويرسل بيانات الى (Server) . لفعل هذا ، استدع الـ PokeData methods (Strings فردية) أو PokeDataLine للـ DdeClientConv الخاص بالعميل . وكل Function ترجع بـ True إذا كانت ناجحة . قم بتمرير اثنين من الـ string arguments الى الـ PokeData : اسم الـ DdeDerverItem الذى تريد أ ن ترسل له البيانات ، والنص الذى تريد إرساله :

```
with DdeClientConv1 do
  if not PokeData('DdeServerItem1', 'I"d rather go sailing')
then
  ShowMessage('Poke failed');
```

استخدم الـ PokeDataLines بنفس الطريقة ، لكن قم بتمرير TStrings أو TStringList على أنه الـ argument الثانية . إن الـ Pokin يعمل فقط إذا تقبل (server) الـ Poked data. فى تطبيقات الـ Delphi، يجب أيضاً أن تنشئ الـ DdeServerItem OnPokeData، الذى يتم اطلاقه عندما يحاول العميل أن يدفع البيانات إلى الـ server المتعاون معه . فى الـ event handler ، تكون البيانات المتاحة من خلال خصائص الـ Lines والـ Text التابعه للـ DdeServerItem .

استخدام تعليمات الـ macros الخاصة بالـ DDE:

إذا كان المساعد يقدم تعليمات macros، فيستطيع تطبيق العميل أن يرسل أوامر macros لأداء العمليات فى الـ Server . على سبيل المثال ، يستطيع تطبيق العميل DDE الخاص بـ Delphi أن يستخدم هذه التقنية لإرسال وتنفيذ أمر macro لوثيقة الـ Microsoft Word .

فى تطبيقات الـ Delphi الخالصة، يرجع تحديد معنى الـ Macro إليك أنت. ولكن إذا لم يكن الـ Server الخاص بك يؤيد الـ Macro، فيمكنك استخدام Components لإرسال واستقبال أوامر الـ Macro. لفعل هذا، يقوم الـ DdeServerConv بتنفيذ للـ OnExcutemacro. عندما يقيحدث هذا الـ event، يجب أن يقوم الـ Server الخاص بك بتنفيذ أوامر الـ Macro فى خاصية الـ DdeSererItem Lines. (بالرغم من أنه بإمكانك استخدام خاصية الـ Text أيضاً، فإنك غالباً ماتستخدم الـ Lines، لأن الـ Macro غالباً مايتكون من سطور متعددة).

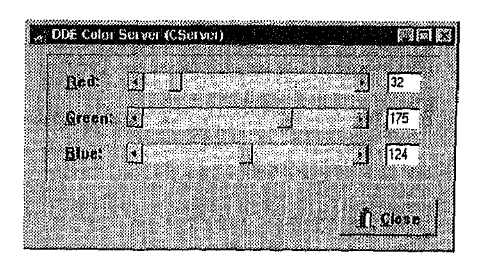
وعلى الوجه الآخر للعملة وهو مايخص العميل استدع الـ ExecuteMacro أو الـ ExecuteMacroLines للـ DdeClientConv. أنك عادةً تستخدم الـ ExcuteMacroLines لإرسال أمر Macro متعدد الأسطر الى الـ Server ليقوم بتنفيذه. إذا كان هذا الأمر موجود فى TStringList يدعى الـ Macro، استخدم هذا الـ Code لإرساله الى الـ Server :

```
with DdeClientConv1 do
  if not ExecuteMacroLines(Macro, True) then
    ShowMessage('Macro failed');
```

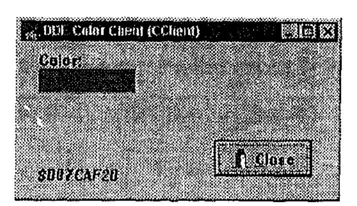
إن الـ ExcuteMacro والـ ExcuteMacroLines ترجع بـ True إذا كانا ناجحين. إن الـ argument الأولى للـ ExcuteMacro هى الـ PChar تشير الى الـ String، وهى TStrings أو TStringList لـ ExcuteMacroLines. وتكون الـ argument الثانية True إذا أردت للبرنامج أن ينتظر الـ Server حتى ينفذ أمر الـ macro؛ حدد الـ argument الثانية بـ False لتمكن العميل من الاستمرار بينما الـ Server يقوم به مع الـ macro الذى تم إرساله.

اختبار برنامج DDE :

على القرص المدمج : كمثال أخير على محادثات الـ DDE، يحتوى دليل الـ DdeColor على القرص المدمج على تطبيقين، الـ CServer، والـ CClient، يرسلان ويستقبلان معلومات لون عبر محادثة DDE. يوضح شكل (١٦-٤) عرض الـ CServer. يوضح شكل (١٦-٥) نافذة الـ CClient.



شكل (١٦-٤) يرسل الـ CServer معلومات لون الى أى من تطبيقات العميل التى تقيم محادثة DDE مع الـ CServer



شكل (١٦-٥) يبدأ الـ CClient محادثة مع الـ Server ويتلقى بصورة آلية كعلومات لون عبر الـ Link

لتشغيل البرامج، اتبع الخطوات التالية :

١- انتقل الى نسخة من دليل الـ Siurce\DdeColor على القرص المدمج .

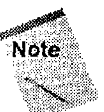
٢- قم بتحميل ملف مشروع الـ CServer.dpr فى Delphi .

٣- اضغط Ctrl+F9 للـ compile، ولكن ليس لتشغيل البرنامج .

٤- قم بتحميل الـ CClient.dpr من نفس الدليل، ومرة أخرى اضغط Ctrl+F9 للـ compile.

٥- اغلق كل الملفات فى Delphi.

٦- باستخدام Windows Explorer، قم بتشغيل الـ CClient.exe. هذا أيضاً يؤدى إلى بدء CServer، إلا إذا كان عاملاً بالفعل (قم بتصغير نوافذ أخرى إذا أصبحت نافذة البرنامج غير مرئية). استخدم scroll bar بال server لتحديث اللون فى تطبيق العميل.



ملحوظة: للحصول على أفضل النتائج، قم دائماً بتشغيل تطبيقات الـ DDE باستخدام الـ Explorer. يمكنك تشغيل برامج DDE من داخل

DDE ، ولكننى واجهت مشاكل فى اقامة روابط الاتصال فى بعض الأحيان . قبل إهدار وقـتك فى إزالة الأخطاء من برامـجك ، حـاول تشـغـيلهـا من خـارج Delphi . (إن الوضع يجب أن يكون قـد تحـسن كـثـيـراً فى Delphi4 عنه فى Delphi2 ، وقد لا تواجه هذه المشكلات) .

إن المساعد المثالى له ثلاثة scroll bars التى يمكنك تعديلها لانشاء قيمة لون مكونة من قيم الـ Byte الأزرق والأخضر والأحمر من صفر الى ٢٥٥ . الـ server يجعل اللون المختار متاحاً إلى أى تطبيقات عميل تقيم محادثات مع server . ويتلقى العـميل المثال بصورة تلقائية قيم اللون الخاصة لـ server ، والتى يمثلها البرنامج فى Edit ثم ابطاله . ويعرض Label قيمة اللون فى كسر عشرى . إن الألوان التى تتـوافق مع الألوان المعـيـارية لـ Delphi ، مـثل الـ cl_White والـ cl_Fuschia ، يتم عرضها بالاسم .

والإخـتـلاف الملحـوظ فى تطبـيق الـ SCerver هو اسـتـخـدام الـ DdeServerConv لبدء محادثة . استخدم هذا الـ Object لتحديد DDE Topic يكون غير مساوياً للتعليق الخاص بنافذة Server . فى الـ CServer ، مثلاً ، يسمى الـ ColorServer.DdeServerConv والـ DdeerverItem المقـابل يحـدد الـ ColorServer فى خـاصـيـة الـ ServerConv التـابعـة لهـ- هذا يخـبر عنصـر الـ Server كيف يقوم بإرسال بيانات خلال محادثة . فى تطبيق الـ CClient ، تكون خاصية الـ DdeTopic للـ DdeClientConv1 محددة أيضاً لـ ColorServer .

على القرص المدمج : توضح القائمة (١٦-٨) كيف يتلقى تطبيق الـ CClient من الـ Server عنصر نص يمثل قيمة لون بالكسر العشرى . هذه الجـــزئيـــة هى من تطبـــيق الـ CClient فى دليل الـ Source\DdeColor على القرص المدمج . ولكن تحويل هذا النص إلى لون ثنائى بطريقة آمنة أمر يتطلب برمجة حذرة .

القائمة (١٦-٨) تطبيق الـ CClient يتلقى بند نص من الـ Server

```
procedure TMainForm.DdeClientItem1Change(Sender: TObject);
var
  S: String;
```



```
   C: TColor;
begin
  with DdeClientItem1 do
     begin
       if Length(Text) = 0 then
           S := '$0'     { Default string }
          else
        S := Text;  { String from server }
          try
       { Delete trailing blanks }
        while S[Length(S)] = asciiBlank do
            System.Delete(S, Length(S), 1);
           C := StringToColor(S);              { Convert to
       color }
           ColorValueLabel.Caption := S;      { Assign to
       label }
           Edit1.Color := C;                  { Show color }
         except
        ShowMessage('Bad color format from server');
         end;
        end;
   end;
```

قـد يكون النص الذى تم اسـتـقـبـاله بلا طول، وفى هذه الحـالة يحـددالـ DdeClientItem1 OnChange للمتغير النتيجة الافتراضية ($0) (فى الـ Object Pascal، تكون قيم الـ strings العشرية مسبوقة بعلامـة الدولار). وإن لم يكن كذلك، يعين البرنامج الـText الآتى إلى strings. قد يكون هذا المتغير متـبوعاً بفـراغـات، وهو الأمـر الذى قـد يجـعل الـ StringToColor function لتـوليـد exception. لذلك، تقـوم while loop بازالة اية فـراغـات لاحـقـة. لاحظ ان البرنامج يستدعى Delete مسبوقة لـ System. وهذا ضرورياً لأن الـ Delete هى أيضـاً method فى Delphi components عـديدة. لإسـتـدعـاء string Delete method فى نظام، فانك غالباً تستخدم الصيغة System.Delete كما هو موضح هنا.

Note **ملحوظة:** بعض Servers قد يرسلون control codes في بيانات النصوص الخاصة بهم. حدد خاصية الـ FormatChars بـ True للـ DdeClientConv الخاص بتطبيق العميل. هذا يؤدى الى فصل control code المسافات الخلفية (#8)، الـ (#9) tab، (#10) line feed ، والـ carriage return (#13) عن النص الذى يتم استقباله.

الـ Object Linking and Embedding (OLE):

إن تقنية تشارك البيانات الأكثر تعقيداً والمتاحة فى الـ Windows تعرف بالـ Object Linking and Embedding . أو الـ OLE. والميزة الرئيسية التى تقدمها الـ OLE هى التحول من نظرة مختصة بالتطبيق للحاسوب الى أخرى تتركز على الوثائق. إن مستخدمى الحاسوب يعملون بالبيانات، وتركيز الإهتمام على الوثائق كـ Objects تبدو أكثر طبيعية لكثير من المستخدمين. مع الـ OLE، يستطيع المستخدمين أيضاً أن يجمعوا المعلومات بطرق غير متوقعة. على سبيل المثال، يمكن أن تحتوى وثيقة برنامج معالج كلمات على صور جرافيكية ثم إنشاءها بواسطة برنامج غير معروف لمؤلف البرنامج المعالج للكلمات. والـ OLE تجعل من الممكن على المستخدمين، وليس فقط على مصممى البرامج، أن ينشئوا أنواعاً جديدة من الوثائق غير مفيدة بـ form معينة لتطبيق واحد.

واستخدام آخر للـ OLE هو الوصول الى Object فى تطبيقات أخرى Microsoft word، مثلاً، وهذا ممكن لأن الـ OLE 2 مؤسس على الـ Comon Object Model، أو الـ COM. وهذه الـ objects توفر واجهة تطبيق يستطيع أى تطبيق أن يستخدمها للوصل الى خصائص Methods فى الـ objects، بغض النظر عن اللغة التى استخدمتها لإنشائهم. COM Objects، ولذلك، فإن Objects الـ OLE، تم إنشاءها من طراز ثنائى يوفر Delphi إليه الوصول التام.

وكما هو الحال مع الـ DDE، فإن الـ OLE، فإن الـ OLE يتطلب تطبيق متعاونين - عميل (Server). ويعرف تطبيق العميل بحاوية الـ OLE . ويوفر الـ Server أوامر لتحرير أنواع معينة من البيانات. تتلقى الحاوية objects من الـ OLE server، إما مباشرة، بواسطة الـ Windows Clipboard، أو بسحب واسقاط ملفات من الـ Windows Expolrer. على سبيل المثال، كـ OLE server، فإن

محرر الـ Form موجة سمعية يمكن أن يوفر أوامر لإنشاء وثائق صوتية. تستطيع حاوية الـ OLE أن تخزنها مع النص الذى يصف الصوت. توفر الحاوية أوامر عامة لإنشاء وثائق صوتية نصية جديدة، ولكن الـ OLE server يوفروا الأوامر لإنشاء، تحرير، ورؤية أو تشغيل أجزاء فردية من البيانات.

إنشاء الـOLE objects :

يستطيع التطبيق أن ينشئ OLE object من أى نوع مسجل من خلال الـ Windows registry. على سبيل المثال، يمكنك إنشاء OLE object فى الـ Microsoft Word، ثم تستخدم هذا الـ object لاستدعاء methods تكتب نصاً، تختار Fonts، تطبع صفحات، وتحفظ وثائق فى ملفات قرص. تستطيع تطبيقاتك أن تؤدى أعمالاً مشابهة مع تطبيقات الـ OLE server - الـ Microsoft Excel، مثلاً.

لإنشاء OLE object، استدع Creat eOleObject function فى Delphi ComObj unit أضف تعريف uses لإدخال هذه الـ unit فى unit، غالباً فى قطاع الـ implementation :

```
implementation
uses ComObj;
```



تحذير: إن العديد من المراجع المطبوعة ومراجع الـ online المتوفرة مع Delphi تخبرك خطأً أن تستخدم الـ OleAuto unit. لاستدعاء الـ CreateOleObject، يجب أن تستخدم ComObj كما هو موضح هنا.

الـ ComObj unit يعرف الـ CreateOleObject والـ parameters الخاصة بها كما يلى :

```
function CreateOleObject(const ClassName: string): IDispatch;
```

- **ClassName :** هذا هو اسم الـ class المسجلة - وهو Word.Basic أو Word.Document، على سبيل المثال - الذى تريد أن تنشئ منه الـ OLE object.
- **IDispatch** - إذا كانت ناجحة، تقوم الـ CreateOleOnject بإدخال object من نوع الـ IDispatch، وهو تعريف واجهة تطبيق مؤسس على الـ



IUnkmown، كـلاهمـا يتم تعـريفـهـمـا فى الـ System.pas unit لـ Delphi . إن الـ IDispatch يتم تعيينه الى Variant ، والذى يتم استخدامه حينئـذ لعـمل استـدعـاءات لـ methods وللوصـول الى خـصـائص فى الـ OLE .

إن اثنين من البـرامج النمـوذجـيـة يوضـحـان كـيـفـيـة اسـتـخـدام الـ CreateOleObject . فى كل برنامج، تقـوم الأوامـر بإنشـاء وثيـقـة Microsoft Word ، وادخال نص فى الوثيقة، وطباعتها وحفظها فى ملف doc. على القرص . وهذه الخطوات تحـدد التـقـنيـات الأسـاسـيـة المطلوبة لإنشـاء واسـتـخـدام OLE Objects- قد تستخدم برمجة مشابهة لإخراج نص تطبيق فى وForm وثيقة تكون مـألوفة للمستخدم . على سبيل المثال، يستطيع نظام قاعدة البيانات أن يعد تقارير ويحـفـظهـم فى وثائق Microsoft Word أو Excel ، كل ذلك دون أن يطلب من المستخدمين أن يفتحوا تلك التطبيقات .

ملحوظة : إن السبب فى أن لهذا الفصل اثنين من البرامج المثالية هو أن الـ Microsoft، قـد اسـتـبـدل الـ Word Basic- وهى لغـة تعليـمـات الـ macro القــــديمة للـ Word والـ Excel - بالـ Visual Basic فى الـ Microsoft Office 97 . هـذا يعنـى أن غـــالبـيـــة الأوامــر، والـ macro (أو التـعليـمـات)، والـ OLE code لم تعـد تعـمـل بالـ Word 97 . إن أسس إنشـاء الـ OLE objects لم تتغير، ولكن التفاصيل أصبحت مختلفة تماماً. إذا كان لديك Windows 95 Word أو نسـخـة أقـدم من هذه، فـانظر أوامـر الـ Word Basic باستـخدام online help للبرنامج . إذا كان لديك الـ Word 97 الحديثة، إبدأ الـ Visual Basic Editor باسـتـخدام أمـر الـ ...ToolslMacro، ثم اضـغط زر الـ Object Browser لتـتـمكن من الوصـول الى online help قـابل للبـحث لـلـ Visual Basic .

الـ Word 95 OLE object والنسخ الاقدم:

على القـرص المدمج : إذا كـان لديك Word 95 أو نسـخـة أقـدم، قـم بتحـميل ملف مشروع الـ OleWord1.dpr من على القرص المدمج فى



دليل Source\OleWord1 الى Delphi . (استخدم OleWord2 إذا كان لديك Word 97 أو ماهو أحدث) إبدأ الـ Word الآن إذا أردت أن ترى الوثيقة التى تم إنشاءها . إذا كان الـ Word لايعمل، فإنه يكون قد تم تنفيذه كـ Server صامت . فى هذه الحالة، تكون الوثيقة مازالت تنشأ وتطبع، ولكن يجب أن يكون الـ Word يعمل حتى ترى نص الوثيقة وهى يتم إدخالها . أيضاً، قم بتشغيل طابعتك- إذا لم يكن لديك واحدة فسوف تتلقى رسالة خطأ من النظام العامل (إن هذه غير ضارة، ولكن بإمكانك أن تحذف عبارة ;V.FilePrint لتتجنبها) .

اضغط F9 لتشغيل البرنامج، ثم اضغط زره لإنشاء وثيقة Word، ادخل النص "Hello form Delphi"، اطبع الصفحة، واحفظ الوثيقة على أنها C:\Hold.doc. (هذا هو الاسم الذى استخدمته للملفات المؤقتة) توضح القائمة (١٦-٩) الـ Source code للبرنامج .

القائمة (١٦-٩) OleWord1\Main.pas

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
    Forms, Dialogs, StdCtrls;

type
  TMainForm = class(TForm)
      Button1: TButton;
        procedure Button1Click(Sender: TObject);
       private
      { Private declarations }
       public
      { Public declarations }
       end;

var
    MainForm: TMainForm;
```

```
implementation

uses ComObj;   // Declares the CreateOleObject function

{$R *.DFM}

{ This procedure works with Microsoft Word 95 and
  earlier versions. It does *not* work with the
     newer Word 97, which now uses Visual Basic in
     place of Word Basic. }
procedure TMainForm.Button1Click(Sender: TObject);
var
  V: Variant;
begin
  V := CreateOleObject('Word.Basic');
     V.Insert('Hello from Delphi');
     V.FilePrint;
     V.FileSaveAs('C:\Hold.doc');
end;

end.
```

Caution

تحذير : يجب أن يكون لديك Microsoft Word 95 أو نسخة أقدم مركبة وتعمل لكي تستخدم التطبيق OleWord1 انظر الـ OleWord2 في هذا الباب إذا كان لديك Word 97 أو نسخة أحدث .

إن عرض البرنامج بسيط ، لذا فهو غير موضح هنا . عندما يبدأ البرنامج في التشغيل ، فهو يعرض زراً كبيراً . اضغط هذا الزر لإنشاء ، طباعة ، وحفظ وثيقة الـ Word . إن OnClick للبرنامج يوضح كيفية استخدام الـ CreateOleObject . كما ذكرت ، يجب أن تضيف تعريف الـ uses لتتمكن من الوصول الى هذه الـ function - يمكنك أن تضيف اسم الـ unit الى تعريف الـ uses الرئيسي للـ module في أعلاها ، أو يمكنك إضافة تعريف الـ Uses كما هو موضح في القائمة بعد كلمة الـ implementation الرئيسية للـ module مباشرة .



لحمل نتائج الـ CreateOleObject، قم بتعريف متغير من نوع الـ Variant مثل هذا :

```
var
  V: Variant;
```

إن الـ Variant هو نوع بيانات خاص وهو، فى الأصل، يمكن أن يكون أى نوع object فعلى. على سبيل المثال، يمكن أن يحمل الـ Variant قيمة عدد صحيح، رقم ذى Point، String. ويمكن أيضاً أن يحمل OLE objects- أو لنكون أكثر دقة، IDispatch object مثل الذى تم إدخاله بواسطة الـ CreateOleObject function. إن الـ Variant objects تؤدى التحويل تلقائياً بحيث إذا حمل واحداً عدد صحيح وقمت أنت بتعيينه الى String، يتم تحويل قيمة العدد الصحيح بصورة تلقائية الى نص.

تحذير : قد تبدو الـ Variant وكأنها مسحورة، ومن الطبيعى أن يغريك أن تستخدمها بشكل مكثف. ولكن، أنها كبيرة وبطيئة - فكل Variant له طول ١٦ Byte على الأقل، بغض النظر عن نوعه - والـ Code المرتبطة بعمليات الـ Variant تستغرق وقتاً طويلاً للتشغيل عن الـ Code التى تؤدى على أنواع بيانات أصلية. لاتستخدم الـ Variant إلا إذا كان يجب عليك هذا - لحمل الـ OLE objects، مثلاً.

بعد تعريف Variant، استدع الـ CreateOleObjectوعيّن النتيجة الى الـ Variant . يمكنك عندئذ استدعاء methods والوصول الى خصائص فى الـ OLE object . على سبيل المثال، إن السطرين التاليين ينشئان Word Basic object ثم يستدعيان الـ Insert method لتلك اللغة لإدخال بعض النص فى وثيقة Word :

```
V := CreateOleObject('Word.Basic');
V.Insert('Hello from Delphi');
```

إنك قد تتعصب، كيف عرف الـ Compiler أن الـ Word.Basic لها method يدعى Insert؟ والإجابة هى : إنه لايعرف. عندئذ استخدام الـ Variants، يقوم Delphi Compiler بإرخاء كل قواعد فحص الـ Syntax الخاصة به ويمكنك من كتابة أى شئ بالاشارة الى الـ Variant . يتم Compile هذه العبارة بشكل سليم :

```
V.NoSuchMethod('Any parameter', 123);
```



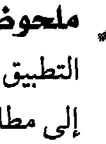
بالرغم من أن الـ Word.Basic ليس لديها هذا الذى يسمى NoSuchMethod. ولكن، عندما تقوم بتشغيل البرنامج، فانك تتلقى رسالة خطأ بهذا المعنى عندما يحاول البرنامج أن يستدعى هذا الـ method فى الـ OLE Word.

وهذه العملية هى مثال على الـ late binding-meaning بمعنى، أنه بدلاً من حل استدعاءات إلى methods و references أخرى أثناء الـ compilation، فإن الحل للـ Variants يحدث فى وقت التشغيل. عندما يستدعى البرنامج الـ CreateOleObject، إذا كان ناجحاً، فإن هذا الـ object يوفر، عبر واجهة التطبيق الخاصة به، قائمة الـ methods والخصائص المتاحة. وهذه هى الوسيلة التى يمكن بها ان تعمل عبارة مثل الـ V.Insert.

وتوضح العبارتان الاخرتان فى OnClick للبرنامج إستدعائين آخرين للـ Word Basic. وهذه العبارات:

```
V.FilePrint;
V.FileSaveAs('C:\Hold.doc');
```

تستدعى الـ FilePrint لطباعة نص الوثيقة والـ FileSaveAs لحفظه فى الملف المشار إليه. إن كل هذه الأعمال تقع فى الـ OLE داخل الـ Microsoft Word.

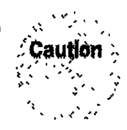


ملحوظة: لايمكنك تشغيل أى تطبيق قديم بانشاء OLE. يجب أن يكون التطبيق OLE server مثل الـ Word أو الـ Excel الذين صمما ليستجيبا إلى مطالب تطبيقات عميل الـ OLE مثل برنامج العرض فى هذا الباب.

OLE object العينة لـ Word 97 وما أحدث:

على القرص المدمج: إذا كان لديك Word 97 أو نسخة أحدث، قم بتحميل ملف مشروع الـ Source\OleWord2 إلى Delphi. إبدأ الـ Word - أحب أن يكون عاملاً حتى يعمل هذا البرنامج بشكل سليم. كذلك، قم بتشغيل طابعتك- إذا لم يكن لديك واحدة، فانك تتلقى رسالة خطأ من نظام التشغيل. (إن هذه غير ضارة) ولكنك يمكنك أن تحذف عبارة الـ V.PrintOut لتتجنبها).

اضغط F9 لتشغيل البرنامج ، ثم اضغط زره لإنشاء وثيقة Word ، إدخل النص "Hello from Delphi," ، اطبع الصفحة ، واحفظ الوثيقة على أنها C:\Hold.doc. توضح القائمة (١٦-١٠) الـ source code للبرنامج .



تحذير : يجب أن يكون لديك Microsoft Word 97 أو نسخة أحدث مركبة وتعمل حتى تستخدم التطبيق OleWord2 أنظر OleWord1 فى هذا الباب إذا كان لديك Word 95 أو نسخة اقدم .

القائمة (١٦-١٠) OleWord2\Main.pas

```
unit Main;

interface

uses
 Windows, Messages, SysUtils, Classes, Graphics,
  Controls,
    Forms, Dialogs, StdCtrls;

type
 TMainForm = class(TForm)
      Button1: TButton;
         procedure Button1Click(Sender: TObject);
        private
   { Private declarations }
     public
       { Public declarations }
     end;

var
 MainForm: TMainForm;

implementation

uses ComObj;  // Declares the CreateOleObject function

{$R *.DFM}

{ This procedure works with the newer Word 97,
 which uses Visual Basic in place of the standard
```

```
   Word Basic as its controlling language. }
procedure TMainForm.Button1Click(Sender: TObject);
var
  V, X: Variant;
    S: String;
begin
  V := CreateOleObject('Word.Document');
    X := V.Range(0, 0);
    X.InsertBefore('Hello from Delphi');
    X.Font.Name := 'Arial';
    X.Font.Size := 18;
    X.InsertParagraphAfter;
    V.Printout;
    V.SaveAs('C:\Hold.doc');
end;

end.
```

إن هذه الطبعة من برنامج OLE تشبه السابقة، ولكنها توضح كيفية استدعاء methods والوصول إلى قيم خصائص فى الـVisual Basic، التى تحل محل الـ Word Basic فى منتجات برمجيات الـ Microsoft. لسوء الحظ، إن اغلب الأوامر مختلفة. لذا فلكى تدعم كلا النسختين من الـ Word، فإنك تحتاج الى تطبيقات منفصلة أو خياراً لاختيار الـ products الصحيح.

إن عملية إنشاء الـ Word 97 OLE object تشبه العملية الخاصة بالـ Word 95، ولكن بدلاً من الـ Word.Basic، فـان الـ CreateOleObject تطلب الآن object من نوع الـ Word.Document. كما سبق، قم لتعريف Variant (وهو V فى الـ procedure، ثم استدع الـ CreateOleObject كهذا.

```
V := CreateOleObject('Word.Document');
```

ان تمرير الـ Word.Document كـ string argument كـوسيط يضمن أحدث نسخة تم تركيبها من الـ Word يتم استخدامها لانشاء الـ OLE object. يمكنك عمل مرجع لنسخة برنامج خاص بربط الرقم الكبير للنسخة الخاصة به. على سبيل المثال، إن هذا أيضاً يعمل:

```
V := CreateOleObject('Word.Document.8');
```



بعد انشاء الـ OLE object ، يمكن للبرنامج ان يقوم باستدعاء methods .
ولكن تذكر، إن الـ methods والخصائص تختلف فى الـ Word Basic عنها فى الـ Visual Basic . والأسوأ من ذلك، إن الخطوات المطلوبة لأداء مختلف العمليات اصبحت الآن مختلفة أيضاً. على سبيل المثال، لإدخال بعض النص الآن فهذا يتطلب على الأقل ثلاثة أوامر مثل:

```
X := V.Range(0, 0);
X.InsertBefore('Hello from Delphi');
...
X.InsertParagraphAfter;
```

إن الاستدعاء للـ Range يمنع استبدال أى نص مختار، والذى لايعد ضرورياً فى هذه الحالة، ولكنه مرشحاً فى دليل برمجة الـ Visual Basic . والزر البيضاوى يوضح أين يمكن ان تذهب الأوامر الأخرى لتغيير جوانب من النص الذى تم ادخاله، والذى تم اختياره طبقاً للنظام الافتراضى. إن البرنامج يستخدم هذا المكان لاختيار اسم font وحجم بالعبارات:

```
X.Font.Name := 'Arial';
X.Font.Size := 18;
```

إن استدعاء الـ InsertParagraphAfter يبدأ فقرة جديدة وهو مساو إلى ضغط مفتاح الـ Enter عند كتابة وثيقة. لطباعة النص الذى تم إدخاله وحفظه فى ملف، يقوم البرنامج بتنفيذ إثنين من أوامر الـ Visual Basic الأخرى:

```
V.Printout;
V.SaveAs('C:\Hold.doc');
```



فكرة : لايوجد متسع هنا لنخوض فى برمجة الـ Visual Basic بالتفصيل. لمزيد من المعلومات، أنظر أفكار برمجة وتراكيب Visual Basic فى online help للـ Word 97 .

استخدام الـ OLEObject:

إن التطبيقين السابقين قد أوضحا كيفية إستدعاء الـ CreateOleObject . وهذه الـ function تنشئ object واحداً، والذى لم يتم بدءه، باستخدام اسم الـ

class والذى يتم تمريره على أنه string parameter . واسم الـ class هذا يجب أن يكون مسجلاً فى الـ Windows registry . احفظ نتيجة الـ function فى Variant . على سبيل المثال، استدع الـfunction مثل هذا لإنشاء وثيقة Word .

```
V := CreateOleObject('Word.Document');
```

إن الـ string Word.Document يعد بالفعل تمثيلاً للـ Class ID (CLSID)، والذى يسمى فى بعض الأحيان بـ GUID (gooey ID). وهذا الـ ID هو hexadecimal key والذى هو مضمون من الناحية الاحصائية أن يكون فريداً. (إذا تداخل أى اثنين من الـ GUIDs ، تكون النتيجة بلا شك فوضى كبيرة). باستخدام الـ Regedit.exe utility فى الـ C:\Windows، يمكنك أن تنظر فى الـ CLSID للبحث عن البرامج المركبة. عندما فعلت هذا، لقد بحثت فى الـ HKEY_CLASSES_ROOT (المدخل الجذرى للـclasses المسجلة ومفاتيحها) عن الـ Word.Document، والذى عرض الـ CLSID التالى :

"{00020906-0000-0000-C000-000000000046}"

هذا هو نفس الـ CLSID للـ Word.Document.8 . وهذا ما يفسر أن أى من التصميمين يعمل لغرض إنشاء الـ OLE . فى المستقبل، إذا قمت بتركيب نسخة أحدث من الـ Word، فاتوقع أن الـ CLSID الخاص بالـ Word.Document سيتم تحديثه فى الـ Windows registry .

إن الـ CreateOleObject function يتم تعريفها فى ملف الـ ComObj.pas التابع لـ Delphi كما يلى :

```
function CreateOleObject(const ClassName: string): IDispatch;
```

وتقوم الـ function بادخال object من نوع الـ IDispatch، والتى تم تعريفها كواجهة تطبيق مشتقة من الـ IUnknown :

```
IDispatch = interface(IUnknown)
    ['{00020400-0000-0000-C000-000000000046}']
    function GetTypeInfoCount(out Count: Integer): HResult;
     stdcall;
      function GetTypeInfo(Index, LocaleID: Integer;
```



```
    out TypeInfo): HResult; stdcall;
      function GetIDsOfNames(const IID: TGUID; Names:
Pointer;
    NameCount, LocaleID: Integer; DispIDs: Pointer): HResult;
    stdcall;
      function Invoke(DispID: Integer; const IID: TGUID;
    LocaleID: Integer; Flags: Word; var Params;
        VarResult, ExcepInfo, ArgErr: Pointer): HResult; stdcall;
  end;
```

إن هذا الـobject يستخدم بعض التراكيب الغريبة والتى لن تجدها فى كثير من الأماكن فى Delphi. إنك لست فى حاجة إلى أن تفهم كل التفاصيل الموجودة هنا: فقط كن على حذر عندما تستدعى الـCreateOleObject، فانك تحصل مرة أخرى على object من نوع الـIDispatch. وهذا الـobject يشير إلى الـOLE، الذى يكمن فى التطبيق المضيف له. إن كل الاستدعاءات لهذا الـobject، عبر واجهة تطبيق OLE object، يتم فعلها باستدعاء function الاستدعاء فى الـIDispatch.

ليس من الضرورى حقاً أن تفهم كل هذه الموضوعات المعقدة المتعلقة بالسجل وقيم الـCLSID. ولكن، قبل الغوص فى بحور برمجة الـOLE، يجب أن تفهم ان الـOLE objects توجد فى برامجها المسجلة وان object تطبيقك هو بالفعل من نوع الـIDispatch. ولأنك لا تستطيع فى أن تعرف حتى وقت التشغيل ما هو واجهة التطبيق التى يوفرها تطبيقك الـOLE، فعليك ان تستخدم Variant لتقوم باستدعاءات OLE object من خلال الـIDispatch. إذا واجهت مشكلة، استخدم Windows Regedit utility لتتصفح اسماء الـobjects المتاحة التى يمكن لتطبيقك أن ينشئها، ولكى تزيل المشكلات. إذا لم يكن الـobject المسجل له CLSID، فلا يمكنك أن تنشئ له OLE، وحتى لو كان له، فلا يوجد ضمان. أن تمرير اسم الـobject class للـCreateOleObject يؤدى إلى نتيجة. (هناك مدخل للـWord.Basic على سجل النظام الخاص بى، ولكن الـCreateOleObject لا يمكنها أن تنشئ OLE من هذا النوع).

Caution **تحذير:** كن على حذر عند استخدام برنامج الـ Regedit . إن التصفح أمر مسموح ، ولكن حتى التغييرات البسيطة Windows registry قد يكون لها نتائج مدمرة . لمزيد من السلامة ، استخدم أمر الـ RegisterylExport Registry File الخاص بالـ Regedit لإنشاء نسخ نص احتياطية للـ Registery باكمله . يمكنك عندئذ استخدام أمر الـ RegistrylImport Registry File لتحميل الـ Registry الذى تم حفظه . انظر أيضاً عنوان مساعدة الـ Regedit وهو "Restoring the registry" لمعرفة الخطوات التى يمكنك استخدامها لاستعادة سجل تم اتلافه .

تقوم الـ CreateOleObject بادخال reference لتعريف واجهة التطبيق والذى يمكن استخدامه للاتصال بالـ object . وللـ CreateOleObject تكون واجهة التطبيق من نوع الـ IDispatch . لانشاء COM object ليس من واجهة تطبيق الـ IDispatch ، استخدم الـ CreateComObject .

كتابة تطبيق حاوية OLE :

إن الفصول السابقة قد أوضحت طريقة واحدة فقط لاستخدام الـ OLE . وهناك طريقة أخرى وهى انشاء تطبيق حاوية يتصل بالـ OLE server . وبهذه التقنية ، يمكنك ربط وتضمين وثائق server فى تطبيقك . على سبيل المثال ، تستطيع حاوية الـ OLE أن تنشئ ، تحمل ، تحرر ، وتحفظ وثيقة الـ Microsoft Excel . ولأن الـ Excel هو تطبيق OLE server مكتمل الصفحات ، يستطيع برنامج Delphi الخاص بك أن يستدعى الـ Excel لإنشاء وثائق ، جميعها من داخل نافذة برنامجك .

إن ربط وتضمين وثائق من أنواع مختلفة- وخاصة إذا كانت تلك الأنواع غير معروفة فى وقت التصميم- هى طريقة رائعة لتمكين المستخدمين من أن يقرروا كيف يتمنون أن يخزنوا المعلومات على حاسباتهم الإلكترونية . يمكنك أيضاً أن تستخدم الربط والتضمين كطريقة لتوفير أوامر معقدة فى برامجك بأقل قدر من البرمجة . بدلاً من قضاء نصف اليوم فى انشاء محرر نص للمستخدمين لادخال وتحرير وثائق ، يمكنك ببساطة أن توفر حاوية OLE وتمكن مستخدمى برنامجك من إنشاء وثائق نص باستخدام المصدر المفضل لديهم - سواء كان هذا Microsoft Word أو WordPerfect .

فى هذا الفـصل، سـوف أوضـح كـيـفـيـة انشـاء تطبـيق حـاوية OLE عـامـة باستـخدام الـ OleContainer من الـ System palette . والبرنامج يوضح كيفية تنفـيـذ الـ in-place editing، والذى بواسطته يتم استـبـدال قـوائم البـرنامج و toolbars الخاص به OLE server . عندما تفتح الـ Excel ، تظهـر الـ toolbars وقوائم الـ Microsoft Excel داخل تطبيق Delphi . وبنفس الطريقة، تفتح وثيقة Word ، تحل أدوات وقـوائم الـ Microsoft Word مـحل واجـهـة التطبيق لتطبيق Delphi . ويوضح البـرنامج أيضـاً كـيـفـيـة تمكين المسـتـخـدمين من تضـمـين وثائق كأيقونات فى التطبيق، بحيث إذا تم فتحها فهى تحضر تطبيقات الـ Server الخاصة بها فى نوافذ منفصلة.

لتوفير تحرير الـ OLE in-place editing، يجب أن يحتوى تطبيق Delphi على :

* OleContainer object واحد لكل OLE ibject فى الحاوية.
* MainMenu object .
* toolbar Panel object (إختيارى).
* statusbar Panel object (إختيارى).

بهذه البنود، يستطيع الـ Server أن يدخل القوائم الخاصة به الـ toolbar والـ SpeedButtons، والـ Status Bar فى الحـاوية. على سـبـيل المثـال، عند تحـرير Microsooft Word متضمن أو مربوط فى حاوية، فإن قوائم الـ Word تزيد على قوائم الحاوية. ويرى المستخدمون أوامر البرنامج المعالج للكلمات والمألوف لهم، ولكنهم لايزالون يقومون بتشغيل الحاوية - فلا يجب عليهم التحول الى أو تشغيل الـ Server باستخدام toolbar Panel خاص بالحاوية، فيجب أن تكون خاصية الـ Align له مـسـاوية لـ alBotton أو alLeft أو alRight، أو alTop، ويجب أن تكون الـ Locked مساوية لـ False. كذلك، يجب أن تستـخدم الحاوية جهاز الـ MDI للتحكم فى الـ Client area بالنافـذة الرئيسـية ولتـوفيـر Child window لتحرير الوثائق. من الممكن أن تستخدم تطبيق احادى النافذة مثل عميل حاوية الـ OLE، ولكن واجهة تطبيق الـ MDI أكثر ملائمة لأنه يمكن المستخدمين من إنشاء وثائق Child window من أى نوع Server .



لإنشــــاء حــــاوية OLE، أضف instance من الـ OleContainer على form . إن الـ object الناتج يشبه الـ Panel ذا سطح مفرغ . وفيه يستطيع المستخدم أن يدخل OLE، والمعروض كأيقونة أو فى Form كاملة .

قم بتجربة هذه الخطوات لتكتب تطبيق حاوية OLE هيكلى يوضح أسس العمل بالـ OLE object :

١- إبدأ تطبيقاً جديداً وأضف OleContainer من لوحة الـ System VCL فى نافذة Form1 .

٢- أضف MainMenu من الـ Standard palette . اضغط مرتين أيقونة القائمة، وادخل قائمتين : Fileو Edit . حدد خاصية الـ GroupIndex لقائمة الـ Edit بواحد. فى قائمة الـ File، ادخل أمر Exit . فى الـ Edit، ادخل أمر IInsert Object.... .

٣- اترك الـ Menu Designer واختر File|Exit من قائمة الـ form . ادخل Close بين الـ begin والـ end فى الـ event handler لأمر القائمة . اختر Edit|Insert Object... من قائمة الـ form، وادخل البرمجة من القائمة (١٦-١١). إن النص مأخوذ من ملف الـ Source\OleCont\Main.pas عيى القرص المدمج .

القائمة (١٦-١١) هذا الإجراء يوضح الخطوات المطلوبة لإدخال OLE فى OleContainer component.

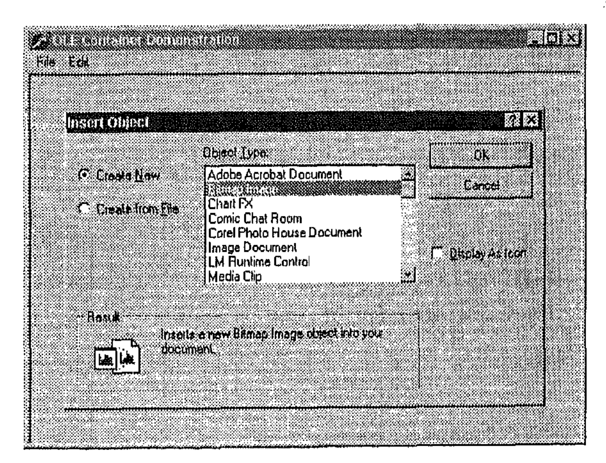
```
procedure TMainForm.InsertObject1Click(Sender: TObject);
begin
    with OleContainer1 do
    begin
  if InsertObjectDialog then
        DoVerb(PrimaryVerb);
 end;
end;
```

إن الـ event handler لأمر الـ ..Edit|Insert Object يوضح الطريقة الأساسية لاستخدام OleContainer . استدع الـ InsertObjectDialog



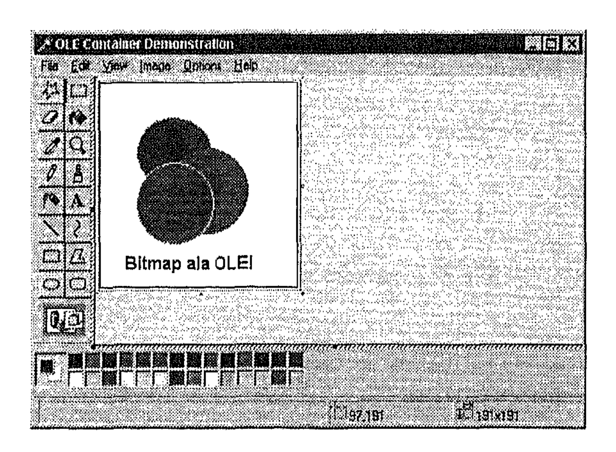
method الخـــاص بالـ object، والذى يعـــرض الـ dialog الموضـح فى شكـل (١٦-٦). يستطيع المستخدم أن يختار نوع الـ OLE لإنشـاء - Bitmap مثلاً- وأيضاً يختارها إذا كان سيدخل صورة كاملة للـ object أو يعرضـه كأيقونة . يمكن للمستخدمين أيضاً أن يختاروا أيقونة لإنتقاء الملف . وكل هذه الأعمـال تقع فى مستوى نظام التشغيل- إن الـ code تقوم باستدعاء الـ InsertObjectDialog لتبدأ العجلة فى الدوران .

إذا رجعت هذه الـ function بـ True ، يكون المستخدم قد اختار object لإنشاءه. لإنهاء العمل وإدخال الـ OLE object فى OleContainer فى تطبيق Delphi، استـدع DoVerb وقم بتمرير الـ argument PrimaryVerb إن الـ DoVerb method يؤدى عـمـلاً للـ OLE object إن الـ PrimaryVerb هو العمل الإفتراضى حسب النظام والوارد لهذا النوع من الـ objects - عادةً مايقوم هذا الفعل بإنشاء وبدء object جديد .

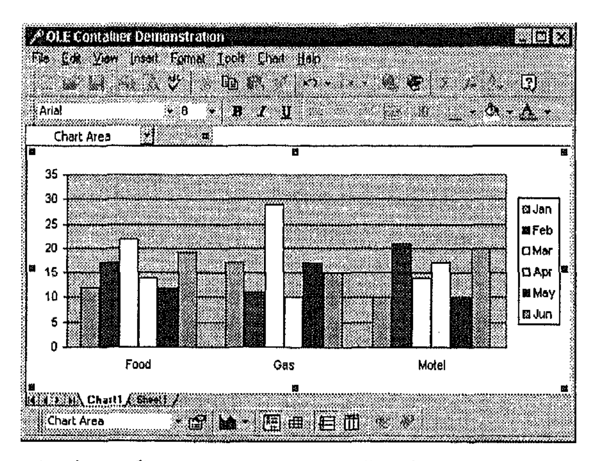


شكل (١٦-٦) الـ Insert Object Dialog معروض بواسطة OleContainer

يوضح الشكل (١٦-٧) تطبيق الـ OleCont بعد ان قمت بانشاء ملف dialog كـاستـجـابة للـ dialog الموضح فى شكل (١٦-٦) . Windows Paint utility - وهى OLE server تحل مـحل قـوائم تطبيق Delphi وتعـرض مع قـوائم التطبـيق الخاص بها . وبقدر يسير من البرمجة ، يوفر البرنامج امكانات تحرير ملف جرافيك كاملة للمستخدمين . يوضح الشكل (١٦-٨) نفس البرنامج ، ولكن هذه المرة عارضاً وثيقة Microsoft Excel Chart . ألا تأمل أن تكون كل البرمجة بهذه السهولة؟



الشكل (١٦-٧) تطبيق الـ OleCont محرراً ملف bitmap باستخدام الـ Windows Paint



الشكل (١٦-٨) نفس تطبيق الـ OleCont ولكن هذه المرة محرراً وثيقة Microsoft Excel Chart.

توضح القائمة (١٦-١٢) البرمجة التى كانت مطلوبة فى Delphi لانشاء وثائق OLE متضمنة بالرغم من أن OleContainer يتم استخدامه بنفس الطريقة، إلا أن البرمجة أصبحت مختلفة للغاية وهى موضحة هنا لأغراض المقارنة فقط. لاتستخدم هذه التقنيات فى التطبيقات الجديدة. انظر القائمة (١٦-١١) لمعرفة الـmethods الصحيحة التى يجب استخدامها لتضمين الـ OLE objects فى تطبيقات حاوية.

القائمة (١٦-١٢) تضمين الـ OLE object
فى OleContainer يتطلب استخدام Code مثل هذه.

```
procedure TForm1.Insertobject1Click(Sender: TObject);
var
 P: Pointer;
begin
 if InsertOLEObjectDlg(Form1, 0, P) then
    begin
      OleContainer1.PInitInfo := P;
         ReleaseOleInitInfo(P);
        end;
end;
```

دمج قائمة الـ OLE:

لتوفير الـ OLE 2.0 in-place editing ، يقوم الـ MainMenu الخاص بتطبيق الحاوية بتعيين قيم معيارية لخصائص الـ GroupIndex لعنصر قائمة عالية المستوى . عندما يقوم المستخدمون بتصوير object ، يحل الـ OLE 2.0 Server محل القوائم ذات الـ GroupIndex المساوية ، ويدخل قوائم إلى اليسار من القوائم التى لها GroupIndex اعلى . ويوفر المساعد أيضاً أوامر الـ Update والـ Exit to Container أو الأوامر المشابهة، عادة فى قائمة الـ File، لحفظ الـ objects الجديدة أو المعدلة فى تطبيق الحاوية .

يذكر الجدول (١٦-٢) الأوامر والقوائم العالية المستوى المقترحة لتطبيق حاوية الـ OLE . حدد الـ GroupIndex الخاصة بعنصر القائمة العالية المستوى بالقيم الموجودة فى الجدول لتمكن من دمج قائمة الـ OLE 2.0 server . على سبيل المثال، حدد الـ GroupIndex الخاصة بالقائمة Edit بواحد حتى يستطيع المساعد أن يستبدل هذه القائمة بالأوامر الخاصة بها .

جدول (١٦-٢) قيم خاصية الـ GroupIndex وقوائم تطبيق حاوية الـ OLE 2.0 المقترحة

الـ GroupIndex	الأوامر المقترحة	القائمة
0	New, Open, Exit	File
1	Insert object, Paste special, Object (set Enabled to false)	Edit

Object	Deactivate	2
View	none	3
Window	Cascade, Tile, Arrange icons	4
Help	none	5

Tip **فكرة :** لا يجب عليك أن تحدد كل قيمة GroupIndex تابعة لعنصر قائمة . قم بتعيين قيم فقط لبنود القائمة العالية المستوى .

قم بتعيين اسم قائمة الـ Object الخاصة بتطبيقك ، بقيمة GroupIndex مساوية لاثنين ، إلى خاصية الـ ObjectMenuItem التابعة للـ form . فى التطبيق الأحادى الوثيقة ، عين هذه الخاصية للـ form الرئيسية . فى تطبيق الـ MDI ، عين الخاصية فى الـ Child form لبرنامجك . بالقيام بهذه التعيينات ، يقوم الـ OleContainer فى الـ form بصورة تلقائية بجعل أوامر تحرير الـ object المختارة .

استخدم قيم الـ GroupIndex بدلاً من تلك الموجودة فى جدول (١٦-٢) لمنع تطبيقات الـ OLE 2.0 server من ان تحل محل قوائم التطبيق الخاصة لك . على سبيل المثال ، إذا قمت بتعيين القيمة تسعة لخصائص الـ GroupIndex لكل قائمة عالية المستوى ، يتم إضافة بنود قائمة الـ server إلى أوامر قائمة التطبيق الخاص بك . لخفض امكانية الخلط (إثنين من قوائم الـ Edit ، مثلاً) ، إذا قمت بهذه الحيلة ، فلا يجب أن يستخدم تطبيقك أى من أسماء القائمة المذكورة فى الجدول .

الـ OLE والـ clipboard :

يجب عليك ان تضيف امر الـ Paste Special لقائمة الـ Edit الخاصة بتطبيقك بحيث يستطيع المستخدمون أن يلصقوا الـ OLE من الـ Windows clipboard . ليس كل تطبيقات OLE server تستطيع نسخ objects إلى الـ clipboard ، ولكن هذا الأمر يمكن المستخدمين من لصق objects للتطبيقات التى تستطيع ذلك . إنك تحتاج إلى أمر الـ Paste Special ، استدع الـ Paste Special Dialog الخاصة بالـ Ole Container . إنك ايضاً عادة ما تريد أن تتأكد إن كانت الحاوية بها وثيقة حالياً ، وفى هذه الحالة يجب حث المستخدمين على التخلص منها . توضح القائمة ١٦-٣ التخطيط العام لأمر PastelSpecial .



القائمة ١٦-٣: نفذ أمر PastelSpecial باستخدام code مثل هذه.

```
procedure TMainForm.PasteSpecial1Click(Sender:
TObject);
begin
  if (OleContainer1.State = osEmpty) or
       (MessageDlg('Delete current OLE object?',
          mtConfirmation, mbOkCancel, 0) = mrOk) then
        if OleContainer1.PasteSpecialDialog then
     begin
      // ... Enable Cut, Copy, and Paste buttons here
        end;
end;
```

هذا الـ code يشبـه تلك الموجـودة فى برنامـج العـرض الخـاص بـ Delphi، وهو OleSdi الموجـود فى دليل Demos\Olectnrs. يتم فـحص خـاصيـة الـ State أولاً لتحديد ما إذا كان الـ OleContainer خالياً. إذا لم يكن خالياً، يقوم dialog رسالة بحث المستخدمين على اذن بحذف الـ OLE الحالى. إذا اجاب المستخدم بـ yes (نعم)، يقوم البرنامج باستدعاء PasteSpecialDialog، والذى يتولى باقى المهام المطلوبة. ويوضح الزر البيضوى أين يمكنك أن تدخل code لتشغيل أزرار أو أوامر قائمة متنوعة الآن حيث قام المستخدم بلصق object فى الحاوية. على سبيل المثال، إذا كان لديك CopyButton، يمكنك تنفيذ هذا الـ code:

```
CopyButton.Enabled := True;
```

كن على حـــذر من أن ليس كل الـ OLE servers يمكنهم لصق objects، وبدلاً من تشغيل زر الـ Paste أوامر قائمة، فمن الأفضل أن تستخدم خاصية الـ CanPaste فى الحاوية. وهذه الخاصية تكون True إذا كان الـ OLE server يمكن من اللصق. قم بتشغيل الـ PasteButton باستخدام code مثل هذه:

```
PasteButton.Enabled := OleContainer1.CanPaste;
```



أفكار للمستخدم الخبير

* فى البرامج التى تلصق objects كبيرة مثل الـ Bitmaps إلى الـ clipboard، فقد تريد أن تمنح المستخدمين فرصة مسح الـ clipboard قبل أن ينتهى البرنامج . وهذا يساعد على توفير الذاكرة . على سبيل المثال، إدخل استفساراً مثل "Clear large bitmap on clipboard?" فى OnClose الخاص بالـ form الرئيسية .

* بالرغم من أن الـ DDE يمكن أن ينقل بيانات النص فقط، فإن ذلك النص قد يمثل قيم عددية، اسماء ملفات، أو هياكل للبايت ممثلة فى hexadecimal strings . وقد يتطلب الأمر بعض البرمجة لاستخدام الـ DDE لنقل الـ objects الثنائية، ولكن إذا عرفت كيف تمثل بياناتك فى الـ text object، فيستطيع الـ DDE حمل المعلومات .

* عند الأتصال بالـ servers الذين لا يستخدمون اسماء الملفات الخاصة بهم مثل الـ Service، فى الـ DdeClientConv لتطبيق العميل، حدد الـ DdeService بالمعّرف الذى تم الحصول عليه من وثائق الـ server. حدد الـ ServiceApplication باسم ملف الـ server . يستطيع العميل عندئذ أن يبدأ الـ server إذا كان الـ ConnectMode، والموجود هو الآخر فى الـ DdeClientConv محدد بـ ddeAutomatic .

* لإضافة scroll bars للـ OLE objec، أضف نموذج من الـ OleContainer فى ScrollBox . لكى تجعل scroll bars مرئية، حدد الخصائص الفرعية Range للـ VertScrollBar و HorzScrollBar بقيم ScrollBox على التوال .

* عند برمجة تطبيقات حاوية الـ OLE الأحادية النافذة، فقد يكون عرض برنامجك أفضل إذا وضعت الـ OleContainer فى Panel . حدد خاصية الـ Align للـ Panel بـ alClient، ثم قم باسقاط OleContainer فى الـ Panel . إنك لاتحتاج أن تقوم بهذا إذا كان تطبيقك يستخدم تصميم الـ MDI المتعدد النوافذ لأنه، فى هذه الحاله، كل Child window يملك OleContainer منفصل .

المشروعات التى يمكنك تجربتها

١٦-١ : اكتب متصفح Clipboard الخاص بك الذى يوفر forms معيارية وكذلك TPicture و TComponent .

١٦-٢ : اكتب تطبيق DDE server الذى ينشئ خطاباً فى الـ Microsoft Word أو أى برنامج معالج كلمات آخر . قد يقدم برنامجك أنواع مختلفة عديدة من الـ forms .

١٦-٣ : اكتب تطبيق عميل DDE الذى يطبع بطاقات أو أسماء أوعناوين تم إدخالها فى الـ Microsoft Excel أو أى تطبيق ورقة حسابات آخر .

١٦-٤ : متقدم: اكتب تطبيق حاوية OLE قراءة فقط والذى يربط اثنين أو أكثر من الوثائق من تطبيقات الـ OLE server على نظامك على سبيل المثال وثيقه معالجة كلمات وورقة عمل ، مثلاً .

ملخص:

* إن الـ Windows clipboard ، تعتبر اسهل طريقة للتشارك فى البيانات فيما بين التطبيقات . تتطلب الـ clipboard التعاون التطوعى لإثنين من التطبيقات (أو اثنين من العمليات فى نفس التطبيق) .

* إن الـ Dynamic Data Exchange، أو الـ DDE، تقدم طريقة اكثر تعقيداً للتشارك فى بيانات نص . لاستخدام الـ DDE، يقوم تطبيق عميل وتطبيق server باقامة محادثة تنتقل عبرها البيانات . تنتقل البيانات عادة من الـ إلى العميل، ولكن التطبيقات تستطيع أن تعكس أدوارها وتمرر البيانات فى الإتجاه الآخر .

* يقوم الـ Object Linking and Embedding، أو الـ OLE، بتحويل التركيز من التطبيق إلى الوثيقة . وكذلك، مع الـ OLE ، يستطيع المستخدمون إنشاء أنواع وثائق جديدة بطرق غير متوقعة من قبل مصممى التطبيقات .

* إن تطبيق حاوية الـ OLE يربط ويضمن الـ objects . يوفر تطبيق الـ OLE server التحرير وأوامر أخرى لأنواع خاصة من الـ objects . استخدم الـ OleContainer التابع لـ Delphi لتنشئ تطبيقات حاوية OLE .

إن كل حاسوب يحتاج إلى نظام قاعدة بيانات جيدة، وكما ستكتشف فى الباب التالى، فإن Delphi يوفر أدوات تطوير قاعدة البيانات التى تعتبر نافعة ومفيدة .



الباب السابع عشر
تطوير تطبيقات قاعدة البيانات

محتويات هذا الكتاب:

- Components.
- تطوير قاعدة البيانات.
- Database components.
- Structured query language.
- Master-detail databases
- استخدام الـData Modules.
- موضوعات برمجة قاعدة البيانات.

بالرغم من أن Delphi يعد أحد أعمدة برمجة الـ Windows، إلا أنه، وكما يوضح هذا الباب، يعتبر مطوراً لقاعدة البيانات أيضاً. مع Delphi، يمكنك إنشاء، تحرير، وكتابة لكل أنواع قواعد بيانات الحاسبات المكتبية تقريباً مثل نظم الـ dBASE، Paradox، والـ ODBC مثل الـ Microsoft Access. يمكنك أيضاً تطوير تطبيقات الـ server والعميل المعقدة . مع Delphi، إنك لن تحتاج أبداً إلى نظام تحكم آخر فى قاعدة البيانات.

فى هذا الباب، سوف أوضح كيفية البدء مع components قاعدة البيانات الخاصة بـ Delphi، وكيف يمكنك استخدامها لإنشاء واستخدام قواعد البيانات فى forms متنوعة. بعد أن تتمكن من بعض الأسس، سوف تتعرف على كيفية أداء أبحاث مع الـ SQL وكيفية إنشاء تطبيقات قواعد بيانات علاقية تعتمد على نمط master-detail table model.

Note

ملحـوظـة: إن كل نسخ Delphi تتـضـمـن الـ Borland Database Engine، أو (BDE)، الذي يوفر مجموعة كاملة من أدوات البرمجة لكثير من نظم قواعد البيانات الحاسبات المكتبية الشهيرة مثل الـ dBase والـ Paradox. تتضمن طبعة الـ Client-Server التابعة لـ Delphi الـ BDE، وهي تمكن من الوصول إلى server قاعدة البيانات مثل الـ Oracle، Sybase، Microsoft SQL Server، Informix. وتأتي طبعة الـ Client-Server من Delphi أيضاً بـ components لتطويل تطبيقات الـ server والعميل. يمكنك استخدام أي طبعة من Delphi مع هذا الباب.

Components:

إن Components قاعدة بيانات Delphi هي:

- **BatchMove** - يؤدي العمليات الجماعية على السجلات والجداول، مثل ازدواج مجموعة بيانات، إلحاق سجلات من احدى مجموعات البيانات إلى أخرى، وتحديث أو حذف سجلات تتماشى مع argument معينة. الـ Palette: Data Access.

- **Database** - يوفر خدمات قاعدة بيانات إضافية مثل server logins والأسماء المحلية. يقوم Delphi بانشاء Database objects بصورة تلقائية على حسب الحاجة، ولكن يمكنك انشاءها بوفرة إذا لزم الأمر. Palette: Data Access.

- **DataSource** - يربط components مجموعة البيانات مثل الـ Table والـ Query بالـ data-aware components مثل الـ DBEdit والـ DBMemo. إن كل تطبيق قاعدة بيانات يحتاج على الأقل واحداً من الـ DataSource objects. Palette: Data Access.

- **DBChart** - وهو component مكتمل الصفات لإنشاء Charts من معلومات قاعدة البيانات. يوضح الباب التالي كيفية استخدام الـ DBChart. Palette: Data Controls.

- **DBCheckBox**- data-aware CheckBox . Palette: Data Controls.
- **DBComboBox**- data-aware ComboBox component . Palette:data Controls.
- **DBCtrlGrid**- مجموعة من scrollablePanels كل منها تمثل سجل قاعدة بيانات واحد. قد تحتوى كل Panels على objects واحد أو أكثر من الـ data-aware control objects. ويستخدم هذا الـ component مع الـ DBNavigator object لتصفح غير تقليدى للبيانات. انظر أيضاً الـ DBGrid. Palette: Data Controls.
- **DBEdit**- component لإدخال نص أحادى السطر data-aware Edit. الـ Palette: Data Controls.
- **DBGrid**- data-aware ، أعمدة وصفوف تعرض سجلات قاعدة بيانات، سجل واحد فى كل صف. يستخدم مع الـ DBNavigator لتصفح البيانات فى جدول تقليدى الـ Palette: Data Controls.
- **DBImage**- data-aware graphical Image يستخدم لعرض الـ Blobs (اشكال ثنائية كبيرة) تحتوى على bitmap . Palette: Data Controls.
- **DBListBox**- data-aware ListBox . Palette: Data Controls.
- **DBLookupComboBox**- data-aware ComboBox له القدرة على البحث فى جدول مراجعة. إن النسخ الأولى من Delphi تسمى هذا بـ DBLookupCombo . Palette: Data Controls.
- **DBLookupListBox**- data-aware ListBox ذا قدرة على بحث جدول مراجعة. إن النسخ الأولى من Delphi تطلق على هذا الـ component اسم DBLookupList . Palette: Data Controls.
- **DBMemo**- data-aware Memo مدخل نص متعدد الأسطر. Palette: Data Controls.
- **DBNavigator**- أداة تحرير وتصفح معقدة لقاعدة البيانات. ويعتبر هذا بالنسبه لبرمجة قاعدة البيانات مثل التحكم عن بعد بالنسبة لجهاز تسجيل الفيديو.

يضغط المستخدمون أزرار الـ DBNavigator للتنقل عبر سجلات قاعدة البيانات، إدخال سجلات جديدة، حذف سجلات، وأداء عمليات أخرى. Palette: Data Controls.

• **DBRadioGroup**- component RadioGroup data-aware.
Palette: Data Controls.

• **DBRichEdit**- مثل الـ DBMemo control، يستطيع الـ DBRichEdit أن يعرض بيانات نص مخزن فى rich-text format (RTF). Palette: Data Controls.

• **DBText**- هى data-aware لعرض معلومات قاعدة البيانات التى لا تريد أن يحررها المستخدمون. (ملحوظة: استخدم الـ Label المعيارى، وليس الـ DBText، لوضع label حقول الادخال على شاشات إدخال البيانات).
Palette: Data Controls.

• **Query**- يحدد عبارات SQL للـ BDE أو للـ SQL server. Palette: Data Access.

• **Session**- تقع كل اتصالات قاعدة البيانات فى موضوعية الـ Session، وهو الذى يتحكم فى هذه الاتصالات. يقوم Delphi بصوره تلقائية بانشاء Session عام لكل تطبيقات قاعدة البيانات، ولكن، يمكنك إضافة Session components لتطبيق ما لتوفر دورات متعددة، والتى قد تفعلها مثلاً للوصول إلى جداول فى مواضع مختلفة من الشبكة. Palette: Data Access.

• **StoredProc**- يمكن التطبيقات من تنفيذ الـ procedures المخزنة على قاعدة بيانات. إذا لم تكن تطور تطبيقات قاعدة بيانات الـ server والعميل، فإنك لن تحتاج غالباً لاستخدام هذا الـ component. تتضمن الـ procedures المخزنة أوامر للحصول على معلومات حول العمليات المغلقة وأرقام الـ IDs لمستخدمى قاعدة البيانات. ولكن، التطبيق الدقيق المتاح يعتمد على الـ server. Palette: Data Access.

- **Table** - يمنح التطبيقات وصولاً إلى قواعد البيانات من خلال الـ BDE . يرتبط هذا الـ component عـادة بالـ DataSource ، الذى يصل الـ Table بالـ data-aware . إن أغلب تطبيقات قاعدة البيانات لها على الأقل Table object واحد . Palette: Data Access .

- **UpdateSQL** - يتوفر هذا للمطوّرين الذين يحتاجون إلى أداء تعديلات على مجموعة بيانات للقراءة فقط التى يتم إدخالها بواسطة SQL server . باستخدام الـ UpdateSQL يمكن أداء أوامر الـ INSERT ، UPDATE وDELETE بالرغم من أن مجموعة البيانات تحمل صفة للقراءة فقط (يمكن أن يحدث هذا ، مثلاً ، عندما يقوم التطبيق بتجميع tables متعددة ، بالرغم من أن الـ tables ذاتها ليست للقراءة فقط) إنك تحتاج هذا الـ component فقط إذا كنت واجهت هذا النوع من المشكلات ، وإن لم تكن ، استخدم الـ Query . Palette: Data Access .

Note

ملحوظة: الـ Data-aware controls تستطيع استخدام معلومات من قاعدة البيانات . على سبيل المثال ، إن الـ DBListBox يشبه الـ ListBox العادى ، ولكنه يستطيع أن يحصل على معلوماته من الـ Table عبر الـ DataSource .

إن النسخ الأولى من Delphi توفر الـ Report فى الـ Data Access palette . وهذا الـ component ، والذى كان يوفر الوصول إلى الـ ReportSmith Report التابعة لـ Borland ، والذى لم يعد متوفراً . لإنشاء تعريفات لقاعدة بيانات ، استخدم الـ components الموجودة على الـ QReport (Quick Report) palette . سيتم شرح الـ QReport (والـ TDBChart) فى الباب القادم .

تطوير قاعدة البيانات:

أن components قاعدة البيانات تجعل الوصول إلى قواعد البيانات أمراً معيارياً وذلك بـ forms متعددة . هذا يعنى أن تطبيقاتك يمكنها الوصول إلى بيانات فى ملفات dBASE ، والـ Paradox Tables ، و الـ Microsoft Access ونظم الـ

Open Database Connectivity (ODBC) الأخرى، أو إذا كان لديك طبعة الـ Client-Server، فمن خلال الـ SQL server.

وافـضـل من ذلك كله، أنه باسـتطاعـتك اسـتـخـدام كل الـ Delphi components الأخرى، لتقنيات واجهة التطبيق، وبرمجة الـ Object Pascal فى تطبيقات قاعدة البيانات الخاصة بك.

استخدام الـ Database Form Wizard:

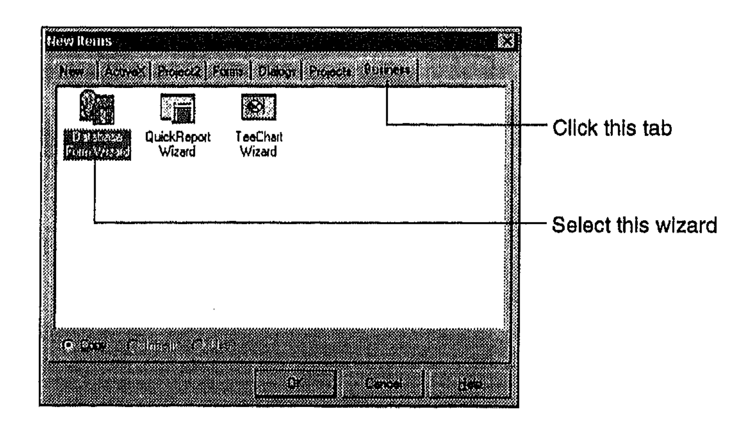
يمكنك تطوير تطبيقات قاعدة البيانات باستخدام أدوات وتقنيات موّضحة من خلال هذا الكتاب. يختلف تطبيق قاعدة البيانات عن برامج الـ Windows فقط فى قدرته على قراءة وكتابة معلومات فى Tables قاعدة البيانات. ومن خلال واجهة تطبيق المستخدم التابع للبرنامج، فإنك تطور form التطبيق كما تفعل مع أى نافذة أخرى.

ولكن، form قـاعـدة البـيـانات النمـوذجـيـة قـد تتطلب edit controls، وlabels، وgrids عـديدة بالإضـافـة إلى الـ components التى توفـر الروابط الضـرورية لـ Tables قاعـدة البيانات. قـد يكون من العسـير أن تبرمـج كل هذه الـ objects على حـدى، فـيـمكنك بدلاً من ذلك أن تسـتـخـدم طبـعـة الـ Database Form Wizard. (ينفـذ من خـلال أمـر الـ Database Form Wizard التـابع لـ Delphi فى قائمة الـ Help فى النسخ الأولى) لتستخدم هذه الـ Wizard الجديدة، إبدأ تطبيـقـاً جـديداً بأمـر الـ ...File | New، اضـغط الـ Business Page tab، واختر الـ Database Form Wizard.

يعـتبـر الـ Database Form Wizard أداة تفـاعلية لإقـامـة forms قـاعـدة البيانات. باختصار، إنك تجيب استفسارات متنوعة وتختار خيارات يعرضها الـ Wizard. عندما تنتهى من ذلك الـ Wizard ينشئ form جـديدة تماماً كـاملة بكل components قـاعـدة البـيـانات فى امـاكنهـا الصـحـيـحـة. يمكنك تحريك هذه الـ components وعـمـل تعـديلات أخـرى للـ form النهـائيـة، ولكن فى كـثـيـر من الحالات، تتطلب النتائج النهائية تعديلاً طفيفاً لإنشاء تطبيقاً تاماً مكتملاً.

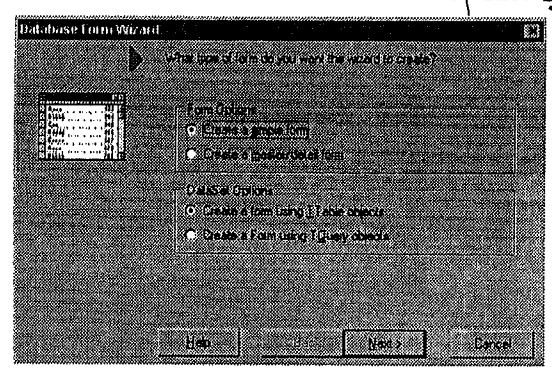
اتبع هذه الخطوات لإنشاء form قاعدة بيانات لـ Tables فى قاعدة بيانات متوفرة مع الـ Delphi (أو يمكنك استخدام Table قاعدة بيانات آخر إذا كان لديك واحداً):

١- اختر ...File | New لتبدأ تطبيقاً جديداً. اختر الـ Business page tab، واضغط مرتين الـ Database Form Wizard (انظر شكل ١٧-١).



شكل ١٧-١: الـ Database Form Wizard على صفحة الـ Business لـ New Items.

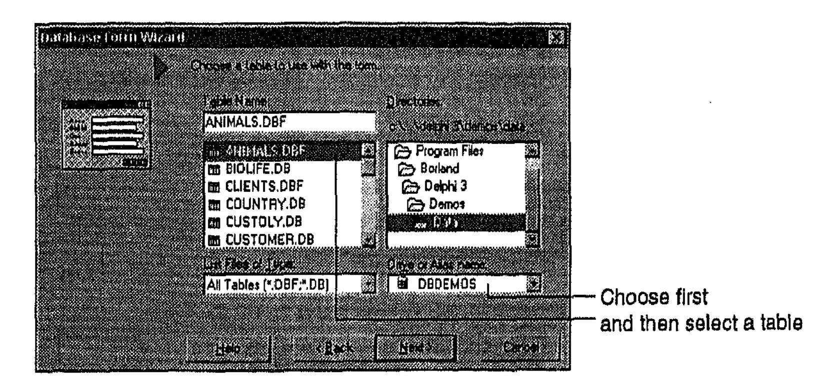
٢- تقدم الـ Wizard صفحات الـ dialog-box، مع خيارات لإنشاء أنواعاً من الـ forms. على الصفحة الأولى، اختر الـ Create a simple form والـ Create a form باستخدام TTable (انظر الشكل ١٧-٢). هذه هي المواصفات الافتراضية حسب النظام.



شكل ١٧-٢: الصفحة الأولى لـ TTable.

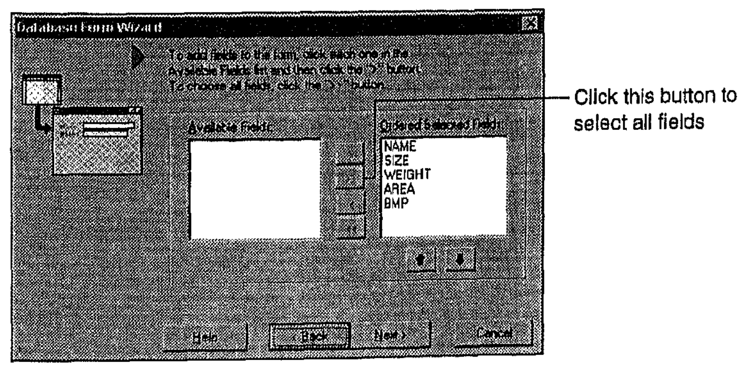
٣- اضغط زر الـ Next لتنتقل إلى الصفحة التالية. (في أي وقت، يمكنك أن تضغط Prev للعودة إلى صفحات سابقة إذا فعلت خطأ ما، أو إذا غيرت رأيك بشأن إختيارات سابقة).

٤- اخـتـر جـدولاً بفـتـح الـ ComboBox الذى يحـمل labeled Drive أو Alias Name . اختر DBDEMOS أو أي اسم قاعدة بيانات إذا كان متاحاً على نظامك . فى قـائمـة الـ Table Name ، يجب أن ترى الآن اسـمـاء الـ Tables التى تكوّن قاعدة بيانات الـ DBDEMOS . اختر ANIMALS.DBF أو Table آخر (انظر شكل ١٧-٣) ثم اضغط Next لتنتقل إلى الصفحة التالية .



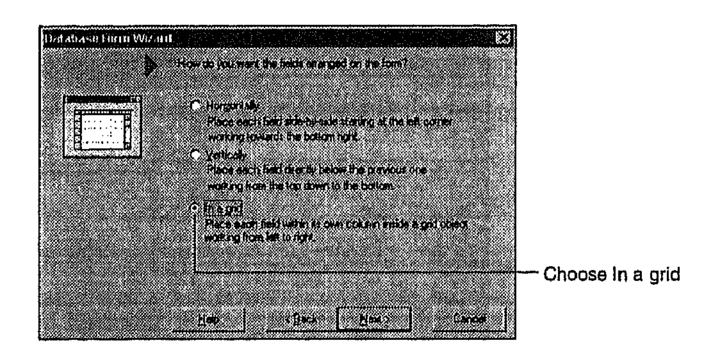
شكل ١٧-٣: اختر اسم قاعدة بيانات واسم الـ Tabl باستخدام الـ Database Form Wizard.

٥- تعطى الـ Wizard الآن قائمة بالمجالات المتاحة فى الجدول المختار. اضغط زر الســهم المزدوج لتنقل كل المجـالات إلى قـــائمـــة الـ Selected، أو يمكنك أن تضـغط Shift+click والـ Ctrl+click للـمـجـالات الفردية واسـتـخدام أزرار الاسـهـم الفـردية (انظر شكل ١٧-٤). اسحب واسقط، أو اضـغط الاسـهـم المشـيـرة إلى أعلى واسفل، لترتيب مجالاتك المختارة فى أى ترتيب، ثم اضغط Next لتنتقل إلى الصفحة التالية.



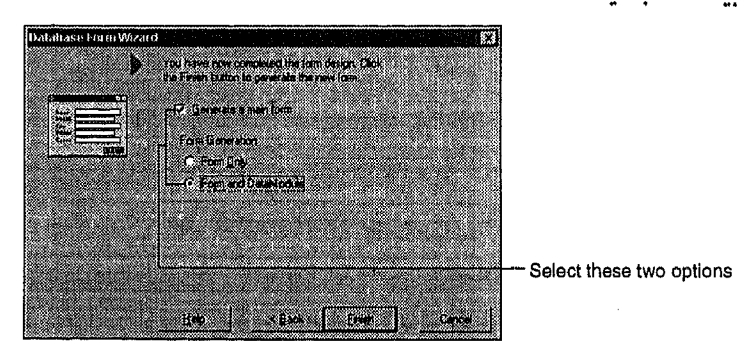
شكل ١٧-٤: اختر المجالات التى تريد أن تستخدمها.

٦- اخـتـر خـيـار تخطيط : Horizontal لـ controls side-By side data-entry وVertical لتضع كل control تحت سابقة ، أو Grid لتنشيء مظهراً يبدو مثل اللوحة لتصفح وتحرير سجلات متعددة ، واحـداً في كل صـف . اختر In a grid (انظر شكل ١٧-٥)، ثم اضغط Next لتنتقل إلى الصفحة التالية .



شكل ١٧-٥: اختر خيار تخطيط مثل الـ In a grid.

٧- إن صـفـحـة Wizard الآخـيـرة تمنحك اخـتـيـار إنشـاء Form بكل الـ components فـقط ، أو Form and DataModule ، بالـ components الغـيـر بصـرية في تركـيـبـة البـيـانات . (هذا يـختـلف عن الـ Wizard الـسـابقين الذين لم يسـتـخـدمـوا data module) . إذا كنت تتابعنا ، تأكـد أن الـ Generate a main form قـد تم اختـيـاره ، واختـر خـيـار الـ Form and DataModule (انظر شكل ١٧-٦) . اضغط زر الـ Finish (كان يطلق عليه Create فيما سبق) ، لتوليد form قاعدة البيانات الجديدة .



شكل ١٧-٦: هذه هي الـ Database Form Wizard dialog page الأخيرة.

ملحـوظة: فى الـ wizard dialog page الأخيـرة، إذا قـمت بإبطال الـ Generate a main form check box، فإنك تحتاج أن تدخل برمجة لعرض الـ form المولّدة- باستدعـاء الـ ShowModal، مثلاً، وباختيار check box، يتم تصميم الـ form الجديدة على أنها الـ form الرئيسية للبرنامج، ويمكنك استخدام الـ Project Manager لحذف الـ form الرئيسية القديمة للمشروع الجديد، والتى تكون خالية ولا تخدم أى هدف.

يمكنك الآن أن تضـغط F9 للـ compile وتشـغيل تطبيق قـاعـدة البيـانات. يوضح شكل ١٧-٧ نافذة البرنامج. بالرغم من أنك لم تضف أى برمجـة، إلا أنه تطبيـق قـاعـدة بيـانات بكافـة وظائفـه! إلى أعلى يوجـد الـ DBNavigator بأزرار لتصفح السجلات، إضافة صفوف جديدة، حذف سجلات، وما إلى ذلك. إنك حر فى استخدام هذه الـ controls، ولكن كن على حذر من أن أى تغييرات تقوم بها يتم تخزينها على الفور فى قاعدة البيانات. على سبيل المثال، قد تغير الـ Area for Angel Fish من الـ Computer Aquariums إلى الـ Tropical Waters.

Form3

NAME	SIZE	WEIGHT	AREA	BMP
Angel Fish	2	2	Computer Aquariums	(BLOB)
Boa	10	8	South America	(BLOB)
Critters	30	20	Screen Savers	(BLOB)
House Cat	10	5	New Orleans	(BLOB)
Ocelot	40	35	Africa and Asia	(BLOB)
Parrot	5	5	South America	(BLOB)
Tetras	2	2	Fish Bowls	(BLOB)

شكل ١٧-٧: تطبيق قاعدة البيانات الذى تم إنشاؤه بواسطة الـ Database Form Wizard المعاون والخطوات المذكورة فى هذا الفصل لاسم قاعدة البيانات DBDEMOS وجدول الـ Animals.dbf table.

إذا قمت بتجربة الخطوات السابقة، فإن إحدى المشكلات التى قد تلحظها هى أن مجالات بيانات الـ bitmap لا يتم عرضها على أنها جرافيك، ولكن بدلاً من ذلك يتم تعريفها على أنها (Binary Large Objects) Blobs. هذا يحدث بسبب الـ DBGrid، إذا اخترت الـ grid التابع للـ Wizard، لا يعرف كيف يعرض الجرافيك. إننا نحل هذه المشكلة فيما بعد بإدخال DBImage فى المشروع. إن الـ Wizard لا تقوم بهذا بصورة تلقائية.

بالرغم من أن الـ DBNavigator يسهل برمجته ، فإن بعض المستخدمين قد يجدونة معقداً فى بادئ الأمر . يأتى مع built-in help hints ، والتى يمكنك تشغيلها بتحديد خاصية الـ ShowHint بـ True . هذا يؤدى إلى تشغيل الـ hints الأصلية ، والتى يمكنك تغييرها إذا لزم الأمر وذلك بتحرير الـ Hints .

يعتبر الـ control DBNavigator مؤثراً ، ويجب أن تكون حذراً من ألا تحذف سجلات أو تقوم بأية تغيرات هائلة ، والتى تصبح دائمة عندما تنقل الحقل الذى يتم إبرازة إلى صف أو سجل آخر . ولكن قبل أن تفعل هذا يمكنك عمل undo لأى تغيير بأختيار زر الـ x . إن ضغط زر علامة الصح يرسل السجل الحالى إلى قاعدة البيانات . وتقوم أزرار الزائد والناقص باضافة وحذف سجلات . والسهم المستدير يجدد البيانات باعادة التحميل من الجدول . وتقوم الأزرار الأخرى بالتصفح عبر سجلات الجدول .

Tip

فكرة : لإدخال forms قاعدة بيانات إضافية فى تطبيق ، اختر أمر الـ Database|Form Wizard . هذا يؤدى إلى تشغيل نفس الـ Wizard الذى اخترته بالـ File|New... فى الـ New Items Business page .

الـ components ومصطلحات قاعدة البيانات:

قبل إنشاء قواعد بيانات وكتابة تطبيقات للتوصل إلى معلومات قواعد البيانات ، فمن المهم أن تفهم ثلاث مصطلحات متعلقة ببعضها عند استخدامها فى برمجة قاعدة البيانات وبواسطة الـ BDE . وهذه المصطلحات هى :

- **Table** – هذا هو مصدر بيانات فى ملف مسطح فردى ، والذى تتصوره على أنه يحتوى على صفوف (سجلات) وأعمدة (مجالات) . وأحد هذه المجالات فى الـ Table هو المفتاح الأساسى الذى يتم فهرسة المعلومات الموجودة فى الملف المسطح داخله . يمكن أيضاً فهرسة الـ Tables فى مفاتيح ثانوية . ويطلق عادة على الـ Table اسم الـ dataset .

- **Query** – مثل الـ Table ، بالرغم من أنه يمكن استخدام الـ Table للتوصل إلى بيانات الـ SQL ، إلا أن استخدام الـ Query يبسط هذه العملية ، وخاصة عندما تأتى البيانات من الـ SQL server . يمكنك أيضاً استخدام الـ Query لإنشاء

إتحادات منطقية من مجموعات البيانات الغير متشابهة (لا تستطيع الـ Tables فعل هذا)، ويمكنك أن تشترك فى بيانات من مصادر مختلفة مثل الـ Paradox Tables والـ SQL.

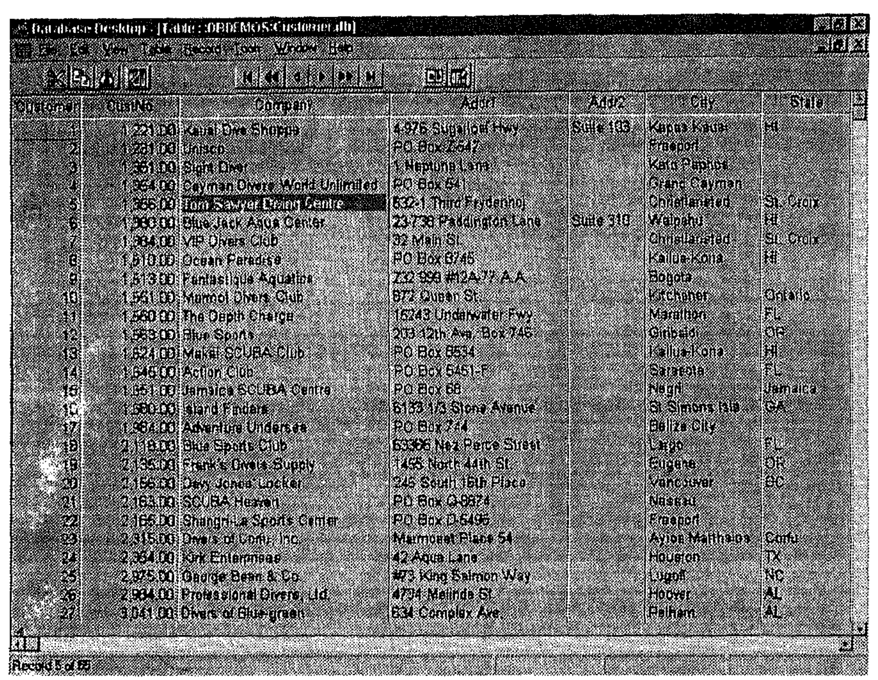
- **Database**- هذه هى مجموعة من Table واحد أو أكثر (اثنين على الأقل دائماً). عندما يشير أحد الـ Tables إلى آخر من خلال قيمة مفتاح فى حقل محدد، وتعرف النتيجة على إنها قاعدة بيانات علاقية.

- **Alias**- هذا هو اسم مسجل مع الـ BDE الذى يخفى اسماء المسارات ومحركات القرص التى تحدد موضع ملفات قاعدة البيانات. استخدم دائماً أسماء مستعارة للإشارة إلى قواعد البيانات، لا تضع ملف Code وأسماء مسارات فى تطبيقاتك باستخدام الأسماء المستعارة، يمكنك أن تنقل ملفات قاعدة البيانات الخاصة بك إلى مواضع أخرى- أو تنقلها إلى دليل على Grid- وتعمل جميع تطبيقاتك بلا تعديل.

إنشاء قاعدة بيانات جديدة:

بالرغم من أنه يمكن استخدام Delphi لكتابة تطبيق يمكنه إنشاء قواعد بيانات جديدة، فان المبرمجين الحاذقين للـ Inprise Corporation قد قاموا بهذا العمل لك فى الـ Database Desktop، وهو متوفر مع Delphi. استخدم هذه المنفعة لإنشاء جداول قاعدة بيانات جديدة، لتعديل البيانات فى قواعد البيانات الموجودة، لرؤية بيانات قاعدة البيانات، ولإنشاء اسماء مستعارة لمصادر البيانات. إن طبعات الـ Professional والـ Client-Server لـ Delphi توفر أيضاً Data Base explorers يمكنك استخدامها لتصفح الـ Tables وهياكلها. إذا كان لديك احدى هذه الطبعات من Delphi، اختر الـ ملزplorers من قائمة الـ Database. يوضح شكل ١٧-٨ الـ Database Desktop وهو يتصفح DBDEMOS Table Customer.db. استخدم الـ Database Desktop، المتوفر مع جميع طبعات Delphi، لإنشاء وتعديل Tables قواعد البيانات فى forms متنوعة. وتوفر أيضاً بعض طبعات Delphi explorers utility قاعدة البيانات مثل الـ SQL Explorer. يوضح شكل ١٧-٩ الـ SQL Explorer وهو يتصفح هيكل نفس الـ Table المستخدم فى شكل ١٧-٨. (يتوفر الـ SQL Explorer فقط مع طبعة الـ

Client-Server من Delphi ، تتضمن طبعات الـ Professional نوع Explorer مشابه ولكن بدون امكانيات الـ SQL) .



Database Desktop - [Table : :DBDEMOS:Customer.db]

File Edit View Table Record Tools Window Help

Customer	CustNo	Company	Addr1	Addr2	City	State
1	1,221.00	Kauai Dive Shoppe	4-976 Sugarloaf Hwy	Suite 103	Kapaa Kauai	HI
2	1,231.00	Unisco	PO Box Z-547		Freeport	
3	1,351.00	Sight Diver	1 Neptune Lane		Kato Paphos	
4	1,354.00	Cayman Divers World Unlimited	PO Box 541		Grand Cayman	
5	1,356.00	Tom Sawyer Diving Centre	632-1 Third Frydenhoj		Christiansted	St. Croix
6	1,380.00	Blue Jack Aqua Center	23-738 Paddington Lane	Suite 310	Waipahu	HI
7	1,384.00	VIP Divers Club	32 Main St.		Christiansted	St. Croix
8	1,510.00	Ocean Paradise	PO Box 8745		Kailua-Kona	HI
9	1,513.00	Fantastique Aquatica	Z32 999 #12A-77 A.A.		Bogota	
10	1,551.00	Marmot Divers Club	872 Queen St.		Kitchener	Ontario
11	1,560.00	The Depth Charge	15243 Underwater Fwy.		Marathon	FL
12	1,563.00	Blue Sports	203 12th Ave. Box 746		Giribaldi	OR
13	1,624.00	Makai SCUBA Club	PO Box 8534		Kailua-Kona	HI
14	1,645.00	Action Club	PO Box 5451-F		Sarasota	FL
15	1,651.00	Jamaica SCUBA Centre	PO Box 68		Negril	Jamaica
16	1,680.00	Island Finders	6133 1/3 Stone Avenue		St Simons Isle	GA
17	1,984.00	Adventure Undersea	PO Box 744		Belize City	
18	2,118.00	Blue Sports Club	63365 Nez Perce Street		Largo	FL
19	2,135.00	Frank's Divers Supply	1455 North 44th St.		Eugene	OR
20	2,156.00	Davy Jones' Locker	246 South 16th Place		Vancouver	BC
21	2,163.00	SCUBA Heaven	PO Box Q-8874		Nassau	
22	2,165.00	Shangri-La Sports Center	PO Box D-5495		Freeport	
23	2,315.00	Divers of Corfu, Inc.	Marmoset Place 54		Ayios Matthaios	Corfu
24	2,354.00	Kirk Enterprises	42 Aqua Lane		Houston	TX
25	2,975.00	George Bean & Co.	#73 King Salmon Way		Lugoff	NC
26	2,984.00	Professional Divers, Ltd.	4734 Melinda St.		Hoover	AL
27	3,041.00	Divers of Blue-green	634 Complex Ave.		Pelham	AL

Record 5 of 55

شكل ١٧-٨: يوضح هذا الشكل الـ Database Desktop وهو يتصفح محتويات ملف الـ DBDEMOS Customer.db المتوفر مع Delphi.

Note

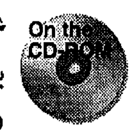
ملحوظة: يعد الـ Database Desktop تطبيق مستقل- يمكنك تشغيلة من قائمة الـ Tools الخاصة بـ Delphi أو من دليل تركيب Delphi . يعتبر الـ SQL Explorer مكملاً لـ Delphi يمكن تشغيله فقط باختيار أمر الـ Database|Explore .

On the CD-ROM

على القرص المدمج : لإنشاء قاعدة بيانات جديدة ، فإن أول خطوة فى تحديد مكان لتخزين معلوماتك ، وتعيين اسم مستعار يشير إلى هذا الموضع . لديك اختيارين : يمكنكم استخدام الملفات الموجودة على القرص المدمج بهذا الكتاب . لاستخدام الملفات ، اتبع الخطوات المرقمة التالية . لإنشاء قاعدة بيانات جديدة خالية ، إنتقل إلى " إنشاء قاعدة بيانات الـ Wines " .

شكل ١٧-٩: هذا الشكل يوضح الـ SQL Explorer وهو يتصفح هيكل ملف الـ DBDEMOS Customer.db المتوفر مع Delphi.

استخدام قاعدة بيانات الـ Wines:



على القرص المدمج: إتبع هذه الخطوات لإستخدام ملفات قاعدة بيـــانات الـ Wines الموجـــودة على القـــرص المدمج فى دليل الـ Source\WinesDemo :

١- انسـخ دليـل الـ Source\Data\Wines ومـلفـــــيّـــــة Wines.px ، وWines.db، على دليل جديد يسمى C:\Database\Wines. يمكنك استخدام اسم مسار وحرف محرك مختلفين، ولكن الدليل الداخلى يجب أن يسمى Wines .

٢- ابدأ الـ Database Desktop باستـخدام الـ Windows Explorer أو Taskbar، أو باختـيـار أمـر الـ Tools|Database Desktop التـابع لـ Delphi عندمـا تظهـر نافـذة الـ Database Desktop، اخـتـر أمـر الـ Tools|Alias Manager.

٣- اضغط زر الـ Next لإنشاء اسم مستعار جديد. إدخل الاسم المستعار WINES ، وتأكـــد من أن الـ Driver Type مـــحـــدد بـ STANDARD (الافتراضى).

٤- اضغط زر الـ Browse، وفى الـ Directory Browser dialog الناتج، اختر حرف المحرك والأدلة لتحديد موضع مسار الـ C:\Database\Wines (أو يمكنك كتابة اسم المسار فى الـ Directories editor). اضـغط OK لتغلق الـ Directory Browser dialog.

٥- اضـــغط Keep New ثم اضـغط OK لإغـــلاق نافـــذة الـ Alias Manager. عندمـا تُسـأل مـا إذا كنت سـوف تحـفظ الـ Public Aliases فى IDAPI32.CFG، أجب بـ Yes.

ملحوظة: إن الفصل القادم يصف كيفية إنشاء قاعدة بيانات Wines خـاليـة وجـديدة. إذا إتبـعت الخطوات السـابقـة، إنتـقل إلى "Components قاعدة البيانات".

إنشاء قاعدة بيانات الـ Wines:

توضح النقاط التالية كيفية إنشاء قاعدة بيانات Wines جديدة وخالية. وهذا يساعدك على معرفة كيفية استخدام الـ Database Desktop لإنشاء Tables قاعدة البيانات الخاصة بك.

إن الخطوة الأولى هى إنشـاء مكان لتخزين الملفـات. إن تحديد اسم الدليل والملف بدقه أمر يرجع إليك، ولكن على نظامى، فقد استخدمت الـ Windows Explorer لإنشاء دليل D:\Database\Wines، (يمكنك استخدام حرف محرك مختلف وكذلك اسم مسار مختلف إذا أردت ذلك). بعد إنشاء الدليل، اتبع هذه الخطوات لإنشاء اسم مستعار لمسار قاعدة البيانات، ولإنشاء ملفات الـ Table:

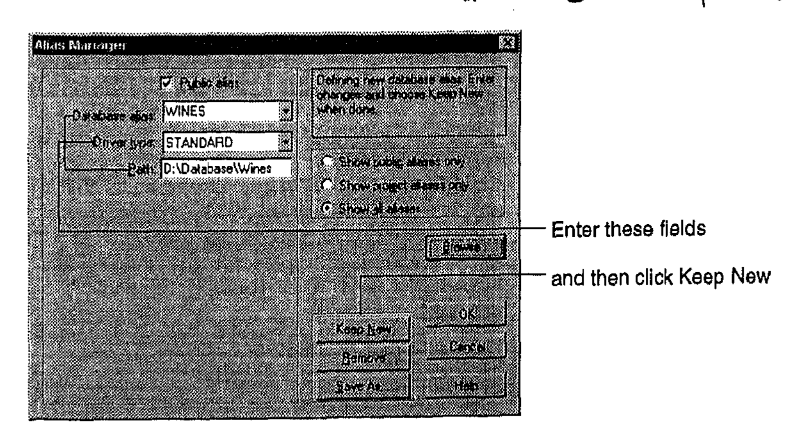
١- إبدأ Database Desktop باستخدام الـ Windows Explorer أو الـ Taskbar، أو باختـيار أمـر الـ Tools|Database Desktop التـابع لـ Delphi. عندمـا تظهـر نافـذة الـ Database Desktop، اختـر أمـر الـ Tools|Alias Manager.

٢- اضغط زر الـ Next لإنشاء اسم مستعار جديد. إدخل الأسم المستعار- يمكن أن يكون string، ولكنه عادة يكون مثل ملف الـ Grid أو الدليل الذى تكون قاعدة البيانات موضوعة فيه. على سبيل المثال، إدخل WINES كاسم مستعار لقاعدة بيانات قد تستخدمه لإقتفاء أثر قائمة جرد مخزن خمور.

٣- حدد الـ Driver Type بـ STANDARD (وهى القيمة الافتراضية حسب النظام). إلا إذا كنت تنشئ قاعدة بيانات الـ server والعميل، ففى هذه الحالة يمكنك اختيار INTRBAS.

٤- اضـــغط زر الـ Brows، واســـحب إلى أســـفـل الـ "Drive or Alias ComboBox control". اختر :D (أو حرف المحرك الذى استخدمته لإنشاء دليل الـ Wines)، ثم اضغط مرتين Database يتبعها Wines فى قـــائمـــة المســـار أعـــلاه. يجب أن يقـــرأ حـــقل الـ Directories هذا "D:\Database\Wines".

٥- اضغط زر الـ OK لإغلاق متصفح الدليل. يوضح شكل ١٧-١٠ عرض الـ Database Desktop فى هذه المرحلة. اضغط OK وأجب بـ Yes عندما تسأل عما إذا كنت سوف تحفظ الـ Public Aliases فى IDAPI32.CFG يمكنك الآن أن تشير إلى اسم الـ WINES OLE فى تطبيقات Delphi بدلاً من محرك الـ code الصلب واسم المسارات فى قاعدة البيانات هذه.



شكل ١٧-١٠: مسار واسم الـ alias WINES مسجلين بواسطة الـ Database Desktop.

إن تسجيل اسم الـWINES OLE وإنشاء مكان لتخزين معلومات قاعدة البيانات . لفعل هذا، إتبع الخطوات التالية ، التى تنشئ قاعدة بيانات فى ملف مسطح لقائمة جرد بسيطة لمخزن خمور :

١ - إبدDatabase Desktop إذا لم يكون عاملاً بالفعل .

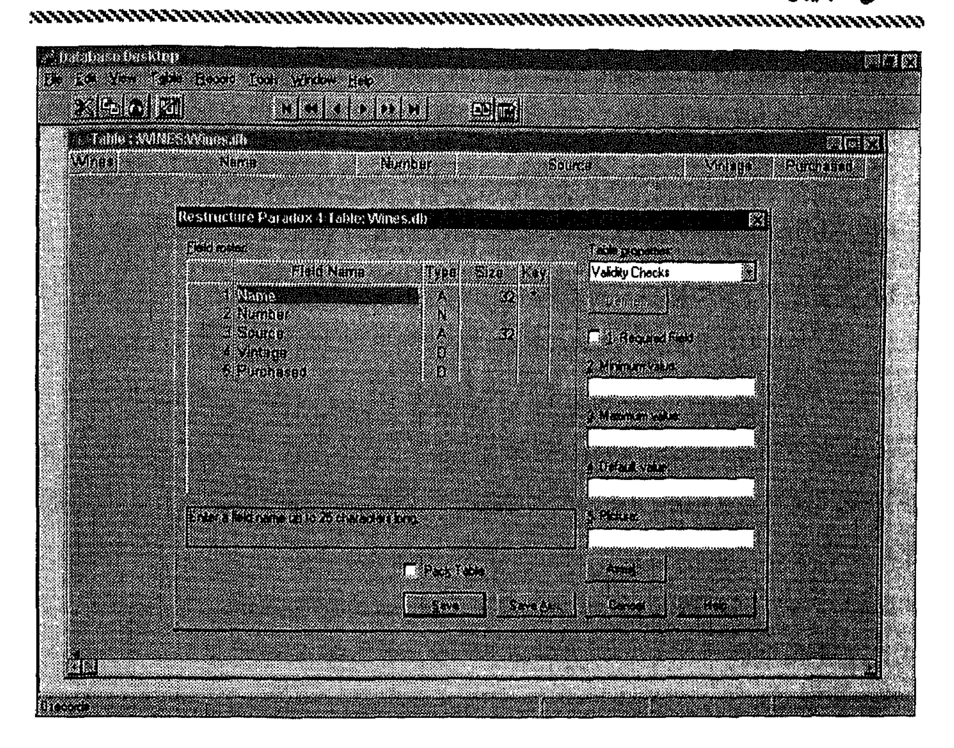
٢- اختر FilelNew واختر أمر القائمة الفرعية ...Table . إنك ترى قائمة forms قاعدة البيانات المتاحة . اختر Paradox 7، وهو الافتراضى . (عند إنشاء قاعدة بيانات جديدة، يعتبر Paradox هو أفضل اختيار- فهو يوفر أغلب أعداد أنواع بيانات المجالات وكذلك إمكانات الفهرسة الرئيسية المعقدة . بالطبع ، يمكنك اختيار أى form قاعدة بيانات أخرى مثل الـ dBASE أو الـ Intrbase إذا أردت) .

٣- إنك ترى الآن شاشة الإدخال الرئيسية لإدخال وتحرير المجالات . إدخل أسماء المجالات ، أنواعها (اضغط قضيب المسافة إذا كانت قائمة) ، وحجم الحقول الأبجدية ، ونجمة (يمكنك كتابة أى رمز) للإشارة إلى حقل Primary key . يوضح الجدول ١٧-١ قائمة ببعض المجالات (هذا هو مجرد عرض، وقاعدة بيانات مخزن الخمور الحقيقية يكون لها مجالات أكثر من هذه بكثير) . يوضح شكل ١٧-١١ هيكل Table قاعدة البيانات التام فى الـ Database Desktop .

٤- بعد إدخال حقولك ، اضغط زر الـ Save As . ادخل Wines فى مربع تحرير Filename، ثم اختر Alias للإشارة إلى Table قاعدة البيانات الجديدة . اختر WINES Alias الذى أنشأته سابقاً (WINES) ، ثم اضغط Save لإنشاء الـ Tables وأى ملفات متعلقة به مثل الفهارس المعتمدة على برنامج تشغيل قاعدة البيانات الذى اخترته فى الخطوة رقم (٢) . إذا كنت مستمر فى المتابعة ، يجب أن يكون لديك الآن ملفين، Wines.db وWines.px فى دليل الـ \Database\Wines .

الـComponents قاعدة البيانات:

بعد إنشاء قاعدة البيانات الجديدة أو اختيار واحدة موجودة بالفعل تكون قد سجلت لها alias ، فأنت مستعد لكتابة تطبيق Delphiلإدخال، تحرير، ورؤية معلومات قاعدة البيانات . كما ذكر، إحدى فى الطرق للبدء هى استخدام الـ Database Form Wizard (استخدم ....FilelNew، اضغط الـ Business page tab، واضغط مرتين الـ Database Form Wizard .



شكل ١٧-١١: هيكل Table قاعدة بيانات
الـ Wines.db التام فى الـ Database Desktop.

جدول (١٧-١) حقول قاعدة بيانات مخزن الخمور العيّنة.

اسم الحقل	النوع	الحجم	المفتاح
Name	Alpha	32	*
Number	Number		
Source	Alpha	32	
Vintage	Date		
Purchased	Date		

ولكن، كما سأوضح فيما بعد، يمكنك إنشاء تطبيقات قاعدة بيانات من لاشئ وذلك بإضافة component على form. فلا يتعين عليك أن تستخدم الـ Wizard. إن أداء هذه الخطوات بنفسك أيضاً يعلمك بما تفعله component قاعدة

البيانات المتنوعة وكيف تتفاعل مع بعضها . حتى إذا خططت لاستخدام الـ Database Form Wizard ، فيجب أن تفهم هذه العلاقات حتى تقوم ببرمجة تطبيقات قاعدة البيانات بنجاح .

Tip **فكرة :** لأن Delphi يولد ملفات متعددة لا تحتاج إلى أن تنشرها على المستخدمين النهائيين مع تطبيقاتك التامة ، فمن الأفضل أن تنشئ أدلة منفصلة لقواعد البيانات وتطبيقاتها .

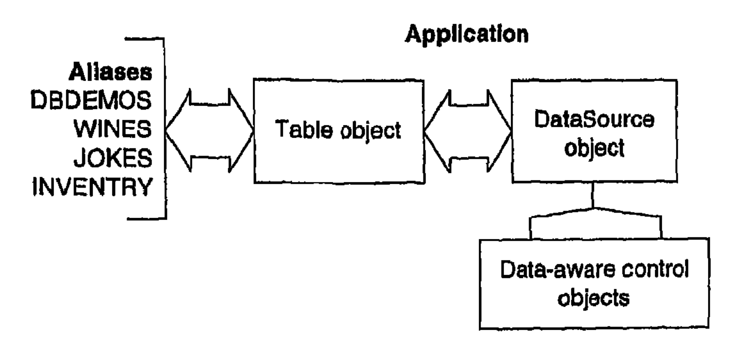
الـData Access components:

إن الـ components الموجودة في الـData Access palette توفر وصولاً إلى Tables قاعدة البيانات . تصور الـData Access على أنها "بوابات" لمعلومات قاعدة البيانات . في أغلب الأحيان ، إنك تحتاج نماذج من Table وDataSource . وتؤدي الـ Data Access الأخرى عمليات SQL مثل (Query) ، وتنفذ عمليات عامة مثل تحديث كل الحقول في سجلات مناسبة (BatchMove) . ويتم استخدام الـDatabase والـ Session بصورة تلقائية عند الحاجة لتنظيم الوصول إلى Tables قاعدة البيانات - إنك لن تحتاج هذه الـ components إلا إذا كنت تنشئ برمجيات متقدمة لأداء مهام مثل الوصول إلى controls إتصال متعددة . إنك لن تحتاج غالباً إلى StoredProc أو الـ Update SQL إلا إذا كنت تطور قواعد بيانات الـ server والعميل .

Note **ملحوظة:** إن النسخ الأولى من Delphi تضمنت برنامج الـ ReportSmith والـReport component لتوليد تقارير قاعدة بيانات . ولقد تم استبدال هذه البنود بـ components توليد التقارير الواسعة النطاق على لوحة الـ QReport . انظر الباب التالي لمزيد من المعلومات عن هذه الـ components وعن الـTDBChart أيضاً .

إن أول component تدخله في form يكون عادة هو الـTable . وهذا الـ object ينشئ كوبري بين التطبيق وalias قاعدة البيانات كما هو موضح في شكل ١٧-١٢ . بالإضافة إلى الـTable ، فإنك تحتاج إلى الـDataSource ، الذي يربط الـ data-aware إلى قاعدة البيانات . إن الـ DataSource يغذي بالبيانات من وإلى

objects أخـرى والـ Table . ويتـولى الـ Table التبـادلات الفـعليـة لقـاعـدة البيانات .



شكل ١٧-١٢: يكون الـ Table كوبرى بين التطبيق وقاعدة البيانات، والمعرفة باسم alias مسجل. يربط الـ DataSource بين الـ Table والـ data-aware مثل الـ DBNavigator والـ DBEdit.

وهذه الأعمـال بين الـ Table، والـ DataSource، والـ data-aware، تقع فى الـ Borland Database Engine (BDE)، الذى يؤدى العمل الحقيقى من قراءة وكتابة بيانات فى الـ form المناسبة . ويمكن للـ data-aware أن تربط . إضافياً لإنشاء شاشات إدخال بيانات متفاعلة . على سبيل المثال ، إن ربط DBNavigator بالـ DataSource، والذى هو مـربوط بالـ Table، يؤدى إلي إنشـاء toolbar متصفح يمكنك استخدامه لرؤية ، تحرير ، ادخال ، وحذف سجلات معروضة فى الـDBEdit ونوافذ control أخرى . وإلى أن تعتاد على أى الـ objects ترتبط بالـ objects الأخرى ، فقد تبدو هذه العلاقات معقدة بعض الشئ ولكن ، بعد أن تنشئ واحداً أو اثنين من التطبيقات ، فستجد أن شاشات إدخال البيانات تتجانس مع بعضها مثل المكعبات الخاصة بالأطفال .

إتبع هـذه الخطوات لإنشـاء Table، وDataSource، وdata-aware لإدخال ورؤية معلومات فى قاعدة بيانات مخزن الخمور (يمكنك أيضاً استخدام هذه التعليمات مع أى قاعدة بيانات أخرى على نظامك) :

١ - إبدأ تطبيقاً جديداً باختيار الـ Database Form Wizard-I- فقد أنشأت الـ code اللازمة بالطريقة الصعبة .

٢- ادخل Table وDataSource من الـ Data Access palette فى form . يتم عرضهما كأيقونات لا تظهر فى وقت التشغيل . لقداستخدمت الـ Names الافتراضية لـ Delphi لكل الـ objects . فى هذا العرض، ولكن فى برامجك، قد تريد أن تعيّن خصائص Name واصفة للـ objects . (إننى أؤيد استخدام الكلمات Table وSource فى هذه الأسماء . على سبيل المثال، MasterTable، NamesSource تعتبر اسماء objects جيدة) .

Tip **فكرة :** يمكنك وضع components غير بصرية مثل الـ Table والـ DataSources فى تركيبة بيانات بدلاً من نافذة form مرئية . يمكنك أيضاً استخدام data moules لإنشاء تركيبات عامة للتوصل لقاعدة البيانات تستطيع التطبيقات المتعددة التشارك فيها. انظر "استخدام data moules object" فى هذا الباب لمزيد من المعلومات .

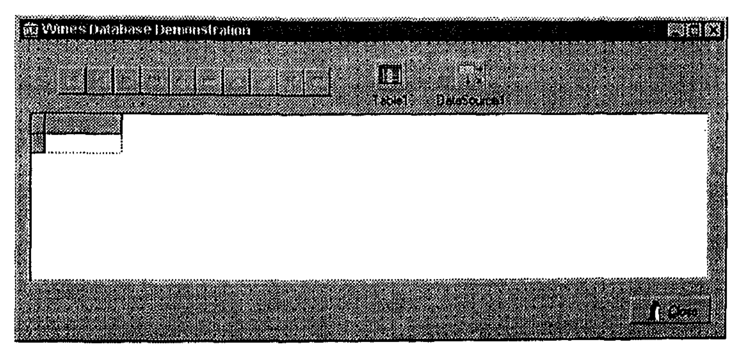
٣- اختر Table1 object، فى الـ Properties الخاصة بالـ Object Inspector، اضغط السهم الواقع بعد خاصية الـ DatabaseNam . هذا يسقط قائمة بأسماء قواعد البيانات وأسماء Alias المسجلة لها . اختر WINES أو اسم آخر وارد فى القائمة .

٤- تذكر أن قاعدة البيانات قد تتكون من Table واحد أو أكثر، وتسمى أيضاً بمجموعات البيانات . بالرغم من أن قاعدة البيانات لمخزن الخمور الخاصة بنا لها Table مجموعة بيانات واحد فقط، بالإضافة إلى اسم Alias للـ DatabaseName، فعليك أيضاً أن تحدد TableName. اختر خاصية الـ TableName للـ Table1، واختر Wines.db أو اسم Table آخر. تذكر : كل Table objects يجب أن تحدد خصائص الـ DatabaseName والـ TableName لها .

٥- بعد ذلك، اختر DataSource1 . اربطه بالـ Table1باختيار Table1 من قائمة اللائحه لخاصية الـ DataSet. بهذه الخطوةو تكون قد أتممت الحد الأدنى للمتطلبات لربط التطبيق بقاعدة بيانات، ويمكنك الآن أن تدخل data-aware فى الـ form لرؤية وتحرير معلومات قواعد البيانات . تذكر : كل الـ DataSource يجب أن ترتبط من خلال خاصية الـ DataSet بـ Table .

٦- وأحد الـ component التى تستخدمها غالباً هو الـ DBNavigator. اختر ذلك الـ component من الـ Data Controls palette، أضف إلى الـ form فى موضع ملائم، فى مكان ما بالقرب من الحد العلوى. خصص الـ DataSource1 لخاصية الـ DataSource التابعة للـ DBNavigator. هذا يخبر الـ data-aware أين تجد بياناتها. ولأن مجموعة البيانات لا تكون مفتوحة فى هذا الوقت، فإن تحديد خاصية الـ DataSource يجعل أزرار الـ DBNavigator غير عاملة. تذكر: كل الـ data-aware يجب أن ترتبط عبر خاصية الـ DataSource بـ DataSource.

٧- لكى توفر سطح لرؤية وتحرير معلومات قاعدة البيانات، أضف BGrid object من الـ Data Controls palette على الـ form. حدد الـ DataSource1 التابعة للـ DBGrid. أعد تحرير حجم وموضع الـ controls والنافذة كما تريد. يوضح شكل ١٧-١٣ نافذة form التطبيق المتطور فى هذه المرحلة فى Delphi.



شكل ١٧-١٣: الـ form المتطورة توضح الحد الأدنى من الـ components المطلوبة لتطبيق قاعدة بيانات نموذجى.

٨- والآن استعد لبعض Wizard. اختر Table1، واضغط مرتين خاصية الـ Active لتغيير قيمتها إلى True. هذا يفتح Table قاعدة بيانات داخل التطبيق المتطور- لا يجب عليك أن تقوم بتشغيل البرنامج. إنك ترى أسماء حقل Table قاعدة البيانات فى الـ DBGrid، وإذا كان Table به أية معلومات، فإنك تراها أيضاً. يعتبر الـ DBGrid الآن active control، وتكون قاعدة البيانات مفتوحة فى هذا الوقت. من الملائم أن يكون لديك بيانات جديدة متاحة عند تصميم تطبيقات قاعدة البيانات- على سبيل المثال، يمكنك بسهولة أكثر أن تجد حجم

components لتعرض معلومات Table - ولكن قد تحدد الـ Active بـ False لإغلاق Table . فى هذه الحالة ، يمكنك إدخال التعيين التالى فى OnCreate الخاص بالـ form (أو فى Procedure آخر) لفتح قاعدة البيانات والـ Table فى وقت التشغيل :

```
Table1.Open;  // Open database and table
```

Caution

تحذير: لا تستطيع الـ Database Desktop أن تعيد هيكلة Table مفتوح . إذا لم تستطيع فتح Table باستخدام الـ Database Desktop ، إرجع إلى Delphi وحدد حقل الـ Active الخاص بالـ Table بـ False . أو ، احفظ وأغلق تطبيقك ، ثم حاول استخدام الـ Database Desktop مرة أخرى .

٩- يوضح شكل ١٧-١٤ تطبيق الـ WinesDemo أخرى .

On the CD-ROM

على القرص المدمج : فى كثير من الحالات ، تستخدم الـ DBGrid لرؤية وتحرير معلومات Table قاعدة بيانات . ولكن هذه ليست هى الطريقة الوحيدة لفعل هذا- يمكنك أيضاً إنشاء forms إدخال بيانات باستخدام العديد من الـ data-aware . على سبيل المثال ، باستخدام الـ DBGrid، يمكنك إنشاء form إدخال قاعدة بيانات توضح سجلاً واحداً فى كل مرة . يوضح شكل ١٧-١٥ ترتيب واحد ممكن للـ control بـ Table الـ Wines.db . يوجد التطبيق التام على القرص المدمج فى دليل Source\WinesEntry - إفتح ملف مشروع الـ WinesEntry.dpr هذا الدليل راضغط برمجة مستخدم ، وهو غير وارد هنا) .

Wines Database Demonstration

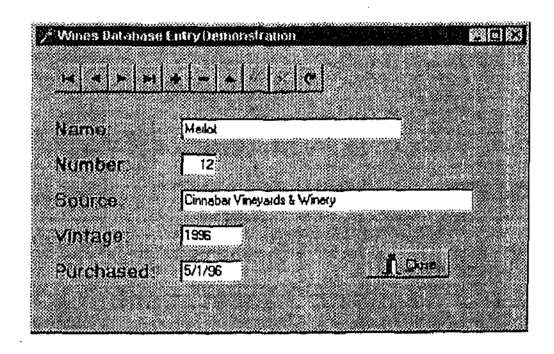
Name	Number	Source	Vintage	Purchased
Chardonnay	8	Mission View Estate Vineyard	1995	6/1/95
Merlot	12	Cinnabar Vineyards & Winery	1996	5/1/96
Sauvignon Blanc	12	Topolos Winery	1996	3/15/97
Cabernet Sauvignon	10	Bates Ranch Santa Cruz Mountains	1992	[illegible]

Close

شكل ١٧-١٤: تطبيق الـ WinesDemo المكتمل.

لإنشاء برنامج الـ WinesEntry الظاهر فى الشكل، فقد أضفت Table و DataSource لـ forms جديدة. ثم اخترت الـ Table، وحددت الـ WINES alias لخاصية الـ DatabaseName وخاصية الـ TableName التابعة له بـ Wines.db. لقد اخترت الـ DataSource وحدد خاصية الـ DataSet التابعة له بـ Table1. بعد هذه الخطوات، فقد أضفت DBNavigator وخمسة DBEdit مع الـ Labels و Button كما هو موضح فى شكل ١٧-١٤. إن حقل الـ DataSource الخاص بالـ DBNavigator والـ DBEdit يكون محدداً بـ DataSource1. تكون خاصية الـ DataField لكل DBEdit محددة باسم حقل مختلف فى جدول الـ Wines.db.

أخيراً، لقد أخترت الـ Table1 وغيرت خاصية الـ Active التابعة له إلى True. ثم ضغطت F9 لتشغيل البرنامج التام.



شكل ١٧-١٥: بدلاً من الـ DBGrid، هذه النسخة من تطبيق قاعدة بيانات الـ Wines تستخدم حقول الـ DBEdit لتحرير وعرض سجلاً واحداً فى المرة.

Data-aware Controls:

عند تصميم forms إدخال البيانات، يجب أن تعرف أى المجالات تريد أن تعرضها وفى أى موضع. لديك إختياران أساسيان: يمكنك استخدام الـ DBGrid لعرض صفوف وأعمدة، أو يمكنك استخدام controls منفرد مثل الـ DBText، DBEdit، و DBMemo لحقول مختارة. يعرض الـ DBGrid كل الحقول المتاحة فى ترتيبها المادى حسب النظام الافتراضى للبرنامج. وإن أمر عرض أى الحقول المنفرد الـ Data-aware الأخرى يرجع إليك.

ولكن، فى كثير من الأحوال إنك تريد أن تحد من الحقول التى تعرضها الـ DBGrid، أو قد تريد أن تغير ترتيب أعمدتها. بالإضافة إلى ذلك، قد تريد أن تنشئ حقلاً أو calculated field. هو حقل وهمى يعرض نتيجة صيغة باستخدام قيم فى خلايا أخرى. يمكنك تنفيذ هذه المهام بتعديل الحقول الفعلية التى يوفرها الـ Table component من المجالات المادية لمجموعة البيانات. يمكن أن يعرض المجال الفعلى جزءاً أو نسخة معدلة من المجال المادى- على سبيل المثال، الاسم الأخير لشخص.

لأداء هذه الأنواع من الأعمال، فبدلاً من استخدام الحقول المادية فى جدول، يمكنك أن تأمر الـ Table أن ينشئ مجموعة من objects المجالات التى فى الحقيقة، تحتوى على أعمدة مجموعة البيانات المادية. تستطيع بعض الأعمدة مباشرة أن تنتقل من العمود المادى إلى الغطاء- يمكنك مثلاً، أن تخبر الـ Table أن يستخدم حقل الـ Name كما هو موجود فى مجموعة البيانات. والأعمدة الأخرى يمكن أن تكون على حسب ما تريده أنت. وهذه الأعمدة "الوهمية" قد تحسب قيم مؤسسة على مجالات أخرى، أو إنها قد تعرض معلومات مهيئة بطريقة خاصة باستخدام البيانات الخام من الـ Table.

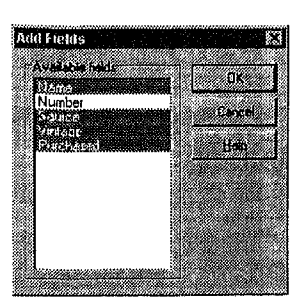
On the CD-ROM

على القرص المدمج : حاول تجربة هذه المفاهيم باستخدام قاعدة بيانات الـ Wines من الفصل السابق- أو يمكنك استخدام Database Form Wizard لإنشاء تطبيق لأى Table مجموعة بيانات. لتحرير حقول الـ Table، اضغط مرتين الـ Table1 فى الـ form.

(إن الملفات الكاملة لهذا الفصل توجد على القرص المدمج فى دليل الـ Source\WinesCalc). كما هو موضح فى الشكلين ١٧-١٦ و١٧-١٧، بالضغط مرتين على الـ Table تعرض محرر الـ Fields الخاص بالـ component والذى يبدأ كنافذة خالية تحمل فى هذه الحالة Mainform.Table1. اضغط يميناً بينما تشير بالمؤشر إلى هذه النافذة. هذا يظهر قائمة ذات أمرين عاملين:

- **Add Fields**- استخدم هذا الأمر لإضافة أعمدة مجموعة بيانات مادية إلى الـ Table. على سبيل المثال، ، إذا أردت استخدام حقول Name كما هو فى ملف مجموعة البيانات، استخدم الـ Add Fields لاختياره. إذا لم تضف حقلاً، فيكون

مخفياً فى المخرجات النهائية- فى الـ DBGrid ، مثلاً . كل حقل مضاف يتم تمثيلة كـ object فى الـ form class . يوضح شكل ١٧-١٦ الـ Add Fields dialog .



شكل ١٧-١٦: الـ Add Fields dialog.

• **New Field**- استخدم هذا الأمر لإنشاء حقول وهمية جديدة ، والتى يمكن أن تكون واحدة من ثلاثة أنواع : Data ، Calculated ، أو Lookup . يتم نقل الـ Data fields من البيانات فى ملف dataset . على سبيل المثال ، يمكنك إنشاء حقل بيانات لعرض uppercase strings فى حروف كبيرة وصغيرة . والـ Calculated fields . تطلق event handle والذى يمكنك استخدامها لأداء الحسابات . (أو أى عمليات أخرى) باستخدام قيم حقول أخرى . فالحقل المحسوب ، مثلاً ، قد يعرض عدد الأيام المنقضية بين تاريخين . والنوع الثالث من الـ New Field الذى يمكنك إنشاؤة يدعى الـ lookup field . هذا يحصل على بيانات من مجموعة بيانات أخرى- اسم شركة ، مثلاً ، يعطى للعميل رقم ID . كل حقل جديد يتم تمثيلة كـ object فى الـ form class . يوضح شكل ١٧-١٧ الـ New Field dialog .

شكل ١٧-١٧: الـ New Field dialog.

لإنشاء برنامج العرض WinesCalc ، الذى يوضح كيفية إنشاء calculated fields ، قمت بالضغط مرتين على الـ Table1 فى نسخة من تطبيق الـ WinesDemo ، ثم استخدمت أمر الـ Add Fields لإضافة أعمدة الـ Name ، والـ Source ، والـ Vintage ، والـ Purchased إلى شكل الـ ١ Table1 . لأننى لم اختر حقل الـ Number فهو مختفى فى الـ Grid النهائية (انظر شكل ١٧-١٦).

ثم استخدمت أمر الـ New Field لإنشاء الـ Days calculated field Old ، من نوع الـ Integer . ففى البرنامج ، يتم تمثيل هذا الحقل على أنه Table1DaysOld object من نوع الـ TIntegerField . (انظر شكل ١٧-١٧) .

Note

ملحوظة: إن التغييرات التى قمت بها فى محرر الـ Fields لا تؤدى إلى تعديل مجموعة البيانات المادية ، لذلك لا تيأس أو تغضب من التجربة . استخدم زر الـ Add لكى تعيد أى حقول تكون قد حذفتها . (جرب هذا : اضغط Clear all ، وأجب بـ yes على الاستفسار الذى يسألك إن كنت تريد أن تحذف كل الحقول . ثم اضغط Add وOK لكى تعيد إضافة الحقول مرة أخرى . كما ترى ، إن المحذوفات لا تكون دائمة) . يتم تعريف كل مجال فى الـ form على أنه component من نوع حقل مجموعة البيانات- TStringField ، TFloatField ، TDateField ، وغيرها . إن استخدام محرر الـ Field يعيد برمجة تعريفات هذه الـ objects .

إن كل حقل فى مجموعة بيانات الـ Table يصبح object فى الـ form class . استخدم الـ Object Inspector لتعديل خصائص هذه الـ objects بنفس الطريقة التى برمجت بها قيم لـ objects أخرى . على سبيل المثال ، اختر Table1 Name فى قائمة لائحة الـ Object Inspector ، وقيم بتغيير خاصية الـ ReadOnly للـ object تصبح True . وعندئذ يصبح المستخدمون غير قادرين على عمل أية تغييرات للقيم فى هذا الحقل .

Tip

فكرة : لاختيار حقل ، اضغط مرتين الـ Table component على الـ form ، واختر الـ object بواسطة اسم العمود الخاص به . أو يمكنك اختيار الـ object باستخدام قائمة لائحة الـ Object Inspector . بأى

الطريقتين، فبعد اختيار الـ object، استخدم نافذة الـ Object Inspector لتعديل خصائص الحقل.

يقوم محرر الـ Field ببرمجة عدد، وأسماء، وأنواع حقول البيانات التى يوفرها الـ Table component للـ Data Source. ما زالت مسئوليتك أن تؤدى الحسابات الفعلية، أو أن تدخل قيم فى أى حقول فعلية تضيفها. إن إضافة حقل باستخدام محرر الـ Fields لا يؤدى إلى إنشاء عمود جديد فى جدول مجموعة البيانات- استخدم الـ Database Desktop لعمل التغييرات للمعلومات المادية الخاصة بالـ Table. يقوم محرر الـ Fields باعادة تشكيل هذه المعلومات للاستخدام فى التطبيق.

على سبيل المثال، بإعطاء مجموعة بيانات لها حقلىّ Date، فقد تريد أن تعرض عدد الأيام الواقعة بينهما. إن تخزين هذه المعلومة فى مجموعة بيانات لا هدف منها لأنها قد تتغير كل يوم لكل تسجيل. هذا هو مثال نموذجى على نوع الحقل الذى يجب أن يحسبه التطبيق. إتبع هذه الخطوات:

١- أعد إضافة DBGrid، إذا لزم الأمر، فى التطبيق وحدد خاصية الـ DataSource له بـ DataSource1. حدد خاصية الـ Active للـ Table1 بـ True لعرض مجموعة البيانات الـ Grid. أعد التعليمات فى الفصل السابق إذا واجهت مشكلة، أو إذا لم تحفظ المشروع.

٢- اضغط مرتين يميناً أثناء الإشارة إلى نافذة المحرر، وأختر أمر الـ Add Field. اختر حقول الـ Name، الـ Source، الـ Vintage، والـ Purchased. (يمكنك استخدام مجموعة بيانات لها على الأقل حقل تاريخ واحد على الأقل). إرجع إلى شكل ١٧-١٦. اضغط OK لإغلاق هذه النافذة.

Tip **فكرة:** إذا أضفت حقل بطريق الخطأ، اضغط يميناً لاظهار القائمة الخاصة بالمحرر واختر Delete لحذف الحقل. هذا يحذف فقط الـ object الذى يمثل الحقل. لا تتغير مجموعة البيانات الحقيقية الموجودة فى الملف.

٣- اضغط يميناً مشيراً داخل محرر الـ Field واختر أمر الـ New Field (راجع شكل ١٧-١٧). ادخل Name مثل Days Old للحقل الجديد، والذى يجب أن يختلف عن أسماء الحقول الأخرى. كما هو واضح فى شكل ١٧-١٧،

فإنك ترى اسم object (Table1 Days Old) فى نافذة التحرير ثانية . هذا هو اسم الـ object كما هو مقرر فى الـ form class- يمكنك تعديل الاسم المولّد بصورة تلقائية فى هذا الوقت، أو يمكنك استخدام الـ Object Inspector فيما بعد لتغيير خاصية الـ object Name .

٤- اختر نوع Field لهذا الحقل الفعلى الجديد . فى هذا المثال، يكون الـ IntegerField ملائماً . وهذا الخيار يحدد نوع البيانات، مثل الـ TStringField والـ TIntegerField ، لتعريف الـ object فى الـ form class . (كل حقل يصبح object فى الـ form) .

٥- تأكد من أن الـ Calculated radio button قد تم اختيارة، ثم اضغط OK لإضافة حقل جديد للـ Table1 . يمكنك عندئذ أن تغلق محرر الـ Field أو تبعد نافذته جانباً . لاحظ أن الـ DBGrid يعرض الآن عموداً جديداً يحمل labeled Days Old . (الـ tool الرأسى إلى اليمين من عمود الـ Name لتقصر هذا الحقل وتجعل الأعمدة الأخرى مرئية إذا لزم الأمر) . اختبر أيضاً تعريف الـ form class ، والذى يتضمن الآن تعريف object للحقل الفعلى الجديد :

```
Table1DaysOld: TIntegerField;
```

٦- إن توفير برمجة تحسب قيمة الحقل الجديد أمر يرجع إليك . افعل هذا باختيار الـ Table1 Days Old باستخدام قائمة لائحة الـ Object Inspector . (لا يمكنك اختيار هذا الـ component فى الـ form لأن حقول مجموعة البيانات ليس لها تمثيل بصرى) . اختر الـ Object Inspector Events page ، ثم اضغط مرتين الـ OnGetText لانشاء event handler خالى . إدخل الـ code التالية فى procedure (الـ procedure بأكملة مذكور هنا كـ reference- إنك تحتاج أن تدخل فقط العبارة الوحيدة الواقعة بين الـ begin والـ end :

```
{ Calculate the Days Old virtual field using today's
  date (returned by the SysUtils Date function) and
     the value of the database table's Purchased value. }
procedure TMainForm.Table1DaysOldGetText(Sender: TField;
     var Text: String; DisplayText: Boolean);
```

```
begin
  Text := FloatToStr(Date - Table1Purchased.Value);
end;
```

٧- اضغط F9 لتشغيل البرنامج وعرض الحقول المحسوبة . يوجد التطبيق WinesCalc التام على القرص المدمج فى دليل Source\WinesCals .

إن الـ Data-aware controls تطلق الـ OnGetText event للحـصـول على مـعلومـات حـقل . فى event handler الـ code السـابقـة ، عـيّن الـ Text parameter البيانات التى تريد عرضها . فى هذا المثال ، يحسب البرنامج عدد الأيام الواقـعـة بين اليـوم وحـقل تاريخ الـ Purchased . إن الـ Date function من الـ SysUtils unit تدخل تاريخ اليـوم على أنه قـيـمـة floating-point . وحـقل الـ Purchased مـذكور من خلال خاصيـة الـ Value للحقل Table1Purchased . هذا هو واحداً من الـ objects التى اضفتها للـ Table باستخدام أمر الـ Add Field التابع لمحرر الـ Field .

ويعتمد نوع خاصية الـ Value للحقل على الـ object- فى هذه الحالة ، للـ TDateField ، تعتبر الـ Value هى نوع بيانات الـ floating-point لحساب عدد الأيام المنقضية بين اليوم وقيمة الحقل ، يقوم البرنامج بطرح قيمتها ويمرر الناتج إلى الـ FloatToStr function التـابعـة لـ Delphi . يتم تعـيين هذه النتـيـجـة الـ Text parameter المتغير ، والذى يتم استخدامة لعرض القيمة المحسوبة فى DBGrid . يوضح شكل ١٧-١٨ البرنامج النهـائى مع عـمـود الـ Days Old المحسـوب فى أقصى يمين Grid قاعدة البيانات .

Tip

فكرة : إن الحـقـول المحسـوبة التى تولد exception قـد تنشئ Loop لا نهـائيـة من رسـائل الخطأ . إذا ظننت أنك لن تسـتطيع التـخلص من الأخطاء- وهذا الأمر شائع لإغلاق dialog الخطأ يؤدى إلى إعادة عرض النافذة ، والتى تعيد حساب الحقل الخطأ ، والذى يولّد exception آخر- إرجع إلى Delphi ، اخـتـر الـ...Tools|Environment Options ، وفى صـفـحـة الـ Debugger للـ dialog الناتج ، اخـتـر Delphi Exceptions واخـتـر الـ User Program تحت عنوان Handled By . (فى Delphi 3 والنسخ السابقة له ، كان

الخيار المساو لهذا هو مربع Break on Exception check box والذى يتم اختياره بواسطة أمر الـ EnvironmentǀTools Options) . ولأنك تقوم بتشغيل البرنامج فى Delphi debugger ، فإن تحديد هذا الخيار يعمل حتى مع التطبيقات التى يتم تنفيذها . هذا يجب أن يؤدى إلى إيقاف الأخطاء اللا نهائية ويعيدك إلى Delphi .

شكل ١٧-١٨: تطبيق الـ WinesCalc النهائى مع عمود الـ Days Old المحسوب.

ويجب عليك أيضاً أن تختار الـ Program Reset (Ctrl+F2) فى قائمة الـ Run . لا تقم بـ reboot حتى تجرب خطوات الاصلاح هذه . ولكن، فى النهاية، قد يكون عليك أن تعيد الـ reboot ، ولكن إنهاء الـ process يجب أن يصلح أى ذاكرة . إذا كنت تستخدم Windows 95 أو Windows98 . إنك غالباً لن تحتاج إلى إعادة الـ reboot مع الـ Windows NT ، والذى يوفر تحكماً أعلى فى الذاكرة . لاحظ أن هذه التقنية تعمل فقط إذا كنت تقوم بتشغيل تطبيق من داخل Delphi .

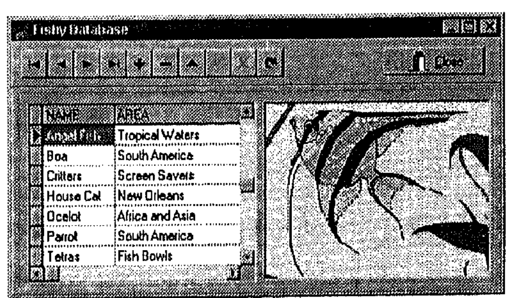
الـ Blob:

إن واحداً من أفلام الرعب القديمة المفضلة لدىّ وهو(The Blob)، وهو يصور وحش بالابيض والاسود لزجة تحوّل ضحاياه إلى بقع على الأرض . وبالنظر إلى تكلفة هذه العملية السينمائية نرى أنها لا تتكلف أكثر من ثمن علبة الجيلاتين وعلبة سائل خفيف . إننى أعرف هذا لأن هذه هى المواد التى كنا نستخدمها لنصنع البقع الخاصة بنا فى الطرقات .

لحسن الحظ، إن حقول الـ Blob الخاصة بـ Delphi ليست بهذه الدرجة من الرعب ولا هى لزجة مثلها . فى حالة أن تكون قد نسيت، فهذه الـ Blob ما هى إلا

إختصار للـ Binary Large Object . فى Table قاعدة بيانات، يستطيع حقل الـ Blob أن يخزن أى نوع من البيانات، ولكنه يستخدم بشكل نموذجى مع الصور الجرافيكية. على سبيل المثال، إن قاعدة بيانات خاصة بأصول عقارات قد تستخدم حقول الـ Blob لتخزين صور عن الأغراض المعروضة للبيع.

على القرص المدمج: إن تطبيق الـ Fishy على القرص المدمج، فى دليل الـ Source\Fishy، يوضح كيفية استخدام مجالات الـ Blob . يوضح شكل ١٧-١٩ عرض تطبيق الـ Fishy. توضح القائمة ١٧-١ الـ source code للبرنامج. لتشغيل البرنامج، يجب أن تكون قد قمت بتركيب ملفات عرض Delphi، وخاصة أولئك على مسار الـ Demos\Data. يجب أن يكون هناك أسماء DBDEMOS مستعارة مسجلة لقاعدة البيانات هذه. إذا لم تستطع أن تجعل تطبيق الـ Fishy يعمل، استخدم الـ Database Desktop كما هو موضح فى هذا الباب لتحدد موضع أسماء الـ DBDEMOS Alias إذا لم تجد هذه الأسماء، أعد تركيب Delphi.



شكل ١٧-١٩: يوضح تطبيق الـ Fishy قدرة Delphi على قراءة وكتابة حقول Blob فى قواعد البيانات. فى هذه الحالة، تكون الـ bitmap Blob موضحة إلى اليمين فى نافذة البرنامج.

القائمة ١٧-١: Fishy\Main.pas.

```
unit Main;

interface

uses
  SysUtils, Windows, Messages, Classes, Graphics,
```

```
    Controls,
     Dialogs, StdCtrls, Forms, DBCtrls, DB, DBGrids,
    DBTables,
     Grids, Buttons, ExtCtrls;

type
  TMainForm = class(TForm)
      ScrollBox: TScrollBox;
        DBNavigator: TDBNavigator;
        Panel1: TPanel;
        DataSource1: TDataSource;
        Panel2: TPanel;
        Table1: TTable;
        DBImage1: TDBImage;
        DBGrid1: TDBGrid;
        Table1NAME: TStringField;
        Table1AREA: TStringField;
        Table1BMP: TBlobField;
        BitBtn1: TBitBtn;
        procedure FormCreate(Sender: TObject);
        procedure DBImage1DblClick(Sender:
      TObject);
     private
      { private declarations }
       public
      { public declarations }
       end;

var
    MainForm: TMainForm;

implementation

{$R *.DFM}

{- Open the Table1 dataset }
```

```
procedure TMainForm.FormCreate(Sender: TObject);
begin
  Table1.Open;
end;

{- Respond to double-click in Blob image }
procedure TMainForm.DBImage1DblClick(Sender: TObject);
begin
  with DBImage1.Picture do
      ShowMessage('W=' + IntToStr(Width) +
          ' H=' + IntToStr(Height));
end;

end.
```

لا يوجد الكثير فى قائمة تطبيق الـ Fishy- فإن أغلب عناصر البرمجة ثم السيطرة عليها من خلال objects الخاصة به. لإنشاء البرنامج، قمت باستخدام الـ Database Form Wizard، وفتحت أسماء الـ DBDEMOS Alias، واخترت ملف الـ Animals.dbf. بعد أن قام الـ Wizard بانشاء الـ form، قمت بإزاحة الـ components جانباً واضفت DBImage لعرض حقل الـ bitmap Blob، والذى يحمل labeled BMP فى جدول قاعدة بيانات الـ Animals. فكرة: لكى تجرّب نفسك فى إنشاء نسخة خاصة بك من هذا البرنامج، حدد خاصية الـ DataSource للـ DBImage بـ DataSource1، وحدد الـ DataField بـ BMP. الباقى معروف وسهل.

لتوضيح كيف يمكنك برمجة events لمعلومات قاعدة البيانات، يقوم برنامج Fishy بتنفيذ الـ DBImage1 OnDblClick event. فلقد استخدمت الـ code الموضحة فى آخر القائمة لتحديد حجم كبر نافذة الـ Blob النهائية (٢٠٠×١٦٠ فى هذه الحالة). قم بتشغيل البرنامج، ثم اضغط مرتين صورة Blob لترى هذه القيم. فى تطبيق أكثر توضيحاً، يمكنك تنفيذ نفس هذا الـ event لتحميل bitmap، أو لتشغيل محرر تشغيل ثنائى لانشاء وتعديل الـ bitmap المخزنة فى قواعد البيانات. استخدم الـ DBImage1.Picture كما هو موضح فى القائمة للوصول إلى الـ bitmap.

الـStructured Query Language:

لقد ظهرت لأول مرة عام ١٩٨٦ بواسطة معهد المعايير الوطنى الأمريكى (ANSI). ولأنه ليس لها هياكل تحكم، فإن الـSQL لا يُعتقد أنها لغة برمجة. ولكن، تقوم الـSQL بتعريف أوامر شبيهة بأوامر البرمجة مثل SELECT، JOIN، وUPDATE لأداء عمليات على Tables قاعدة البيانات. وتعتبر الـSQL قديمة وصعبة الاستخدام، ولكن كل قواعد البيانات تقريباً تقدم على الأقل فرعاً من اللغة، لذا يمكنك الاستفادة من معرفة كيفية استخدامها.

Note

ملحوظة: البعض ينطقون SQL على أنها "سكويل" والآخرون يقولون (إس، كيو، إل). والكثيرون يقولون مصطلحات أخرى لا مجال لذكرها هنا.

لأن Delphi والـBDE يؤديان أغلب خدمات قاعدة البيانات، إلا إذا كنت تطور تطبيقات الـServer والعميل لقواعد بيانات الـRemote SQL، فإنك غالباً تحتاج أن تتعلم أمر SQL واحد وهو: SELECT. كما سأوضح فى هذا الفصل، يمكنك استخدام هذا الأمر لأداءأنواع كثيرة مختلفة من البحوث على المعلومات المخزنة فى جداول قواعد البيانات.

إن الـBDE يؤيد فرعاً من الـSQL المعيارية للـParadox، dBase، Oracle، وقواعد بيانات أخرى. إذا كانت tables قواعد بياناتك ضمن الـform المؤيدة، فيمكنك استخدام أوامر الـSQL الذاتية الوضوح SELECT، INSERT، UPDATE، وDELETE لأداء هذه العمليات. ولبعض الـSQL server الآخرين يوفرون أوامر إضافية- إرجع إلى وثيقة المساعد الخاصة بك لمعرفة الأوامر التى يمكنك استخدامها بالضبط.

الـQuery:

تستخدم تطبيقات الـDatabase تطبيقات الـQuery لإصدار عبارات SQL إلى قاعدة بيانات. إنك تحتاج Query object واحد لكل قاعدة بيانات ترسل إليها بأوامر الـSQL. فى تطبيقات Delphi، إنك غالباً ما تستخدم Query objects لأداء بحوث تعتمد على قضايا مثل "كل الاشخاص الذين لهم أعين بنية والذين ولدوا يوم الثلاثاء وينتمون إلى رابطة البولينج" والمعايير الغربية الأخرى.

إن حـالة من الـ Query component تشبـه الـ Table لأن الـ Query يوفـر منفذاً إلى قاعـدة البيانات من خلال اسم alias مسجل . بالإضافة إلى الـ Query ، فـإنك تحتـاج إلى الـ DataSource لربط الـ Query بالـ data-aware controls . وتوضح الـ controls بشكل نموذجى النتائج أوامر الـ SQL المنفذة .

إنشاء محرر SQL:

على القرص المدمج : إن تطبيق الـ SQLPlay على القرص المدمج ، فى دليل الـ Source\SQLPlay ، يوفر مكاناً للبدء فى إنشـاء محرر لإصدار أوامر الـ SQL إلى قـاعدة بيانات مختـارة . هذا يوضح كيـفيـة إصدار والاستجابة لأوامر الـ SQL باستخدام الـ Query . يوضح البرنامج أيضاً كيفية استجواب الـ BDE عن كل أسماء الـ Tables وأسماء Alias لقاعدة البيانات- إنك تريد أن تفعل هـذا فى تطبيقك لتوفر سبلاً للمستخدمين لاختيار جداول قاعدة البيانات المتاحة .

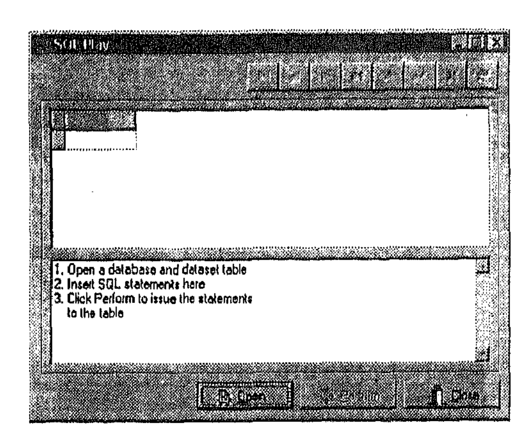
قم بتشغيل تطبيق الـ SQLPlay الآن بتحميل SQLPlay.dpr فى Delphi وضغط F9 . يوضح شكل ١٧-٢٠ النافذة الرئيسية للبرنامج قبل فتح أى جداول قاعدة بيانات .

اضـغط زر الـ Open للـ SQLPlay لعـرض الـ dialog box الموضح فى الشكل ١٧-٢١ . استخدم هذا الـ dialog لاختيار قـاعدة بيانات وجدولاً مثل DBDEMOS وCUSTOMER ، ثم اضـغط OK لتعـود إلى النافـذة الرئيسيـة للبرنامج .

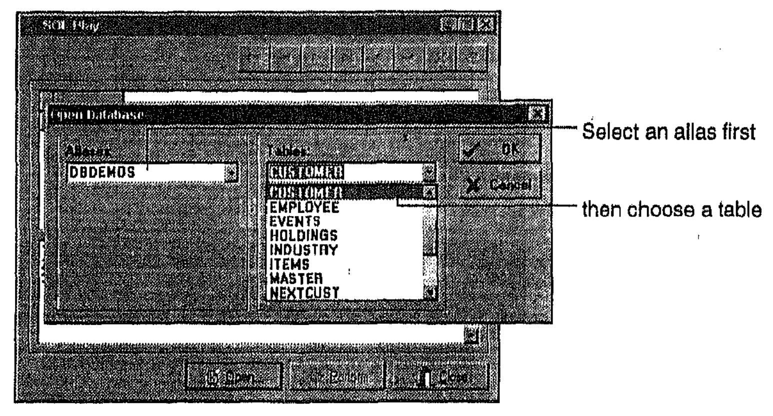
قـم بإدخـال أوامـر SQL فى Memo تحت الـ DBGrid الذى يوضح معلومات قاعدة البيانات . اضـغط Perform لتنفيذ أوامرك . على سبيل المثال ، افتح قاعدة بيانات DBDEMOS وجدول Customer . ثم قم بتحرير أمر نافذة الـ Memo إلى السطور التالية ، والتى تبحث عن كل العملاء فى كاليفورنيا . اضـغط Perform أو اضغط Alt+P لاصدار أمر SQL لجدول قاعدة البيانات :

```
Select * From CUSTOMER
 where State="CA"
```





شكل ١٧-٢٠: هذا هو عرض برنامج الـ SQLPlay قبل فتح جدول قاعدة بيانات.



شكل ١٧-٢١: الـ Open dialog للـ SQLPlay يذكر قائمة بالـ Tables وأسماء الـ Alias لقاعدة البيانات المتاحة.

توضح القائمة ١٧-٢ الـ SQLPlay dialog module، وهي Open.pas . وهذه الـ unit توضح كيفية الحصول على كل أسماء الـ Alias المسجلة لقواعد البيانات واسماء Tables مجموعة البيانات الخاصة بها وكيفية تقديم هذه المعلومات في قائمة لائحة يستطيع المستخدمون أن يختاروا منها Table مجموعة بيانات لتحريره ورؤيته .

القائمة ١٧-٢: SQLPlay\Open.pas.

```
unit Open;
```

```
interface

uses
  Windows, Messages, SysUtils, Classes, Graphics,
   Controls,
    Forms, Dialogs, StdCtrls, Buttons, ExtCtrls, Db,
   DbTables;

type
  TOpenForm = class(TForm)
      OKBtn: TBitBtn;
        CancelBtn: TBitBtn;
        Bevel1: TBevel;
        ComboBox1: TComboBox;
        Label1: TLabel;
        Bevel2: TBevel;
        Label2: TLabel;
        ComboBox2: TComboBox;
        procedure FormActivate(Sender: TObject);
        procedure ComboBox1Change(Sender: TObject);
      private
        { Private declarations }
    public
      { Public declarations }
      end;

var
  OpenForm: TOpenForm;

implementation

{$R *.DFM}

procedure TOpenForm.FormActivate(Sender: TObject);
begin
  Session.GetAliasNames(ComboBox1.Items);
```

```
end;

procedure TOpenForm.ComboBox1Change(Sender: TObject);
begin
 Session.GetTableNames(ComboBox1.Text, '*.*',
     False, False, ComboBox2.Items);
       ComboBox2.ItemIndex := 0;
end;

end.
```

عند تنشيط الـ form (انظر الـ procedure FormActivate، للـ OnActivate event الخاص بهذه الـ form)، تقوم الـ dialog box unit باستدعاء الـ GetAliasNames للحصول على قائمة string بأسماء الـ Alias لقاعدة البيانات من الـ BDE. وهذا الـ method- كثيرين غيره- مزود بـ TSession class لاستخدام هذا الـ object، أضف Db إلى عبارة الـ uses الخاصة بالـ moudle، واستدع الـ TSession methods بالاشارة إلى الـ Session، والذى ينشئه Delphi بصورة تلقائية لكل تطبيقاتت قواعد البيانات. إن الـ GetAliasNames يمسح ويدخل أسماء Alias فى خاصية الـ TStrings أى الـ TStringList. هنا، يستخدم البرنامج الـ method لبدء بنود الـ ComboBox1.

لأن أسماء الـ Table تعتمد على أى أسماء الـ Alias يختار المستخدم، فإن الـ ComboBox الثانى لا يمكن بدءه فى الـ FormActivate. بدلاً من ذلك، فبعد أن يختار المستخدم قاعدة بيانات، فإن الـ ComboBox1Change procedure- والذى يتم اطلاقه عندما تتغير نافذة تحرير الـ control- يستدعى TSession method آخر، وهو GetTableNames، لملء معلومات الـ ComboBox2. يقوم الـ TSession بتعريف الـ GetTableNames والـ parameters الخمسة التابعة لها كما يلى:

```
procedure GetTableNames(const DatabaseName, Pattern: string;
  Extensions, SystemTables: Boolean; List: Tstrings);
```

- **DatabaseName**- هذا هو اسم قاعدة بيانات أو Alias name مثل DBDEMOS.

- **Pattern**– حدد هذا بامتداد اسم الملف مثل db.* لتقصر اسماء الـ Table على الملفات المناسبة .

- **Extensions**– حددها بـ True إذا كنت تريد للأسماء المدخلة أن تتضمن إمتدادات اسم ملف . يكون هذا الـ parameter ذا قيمة لتطبيقات قاعدة البيانات للحاسب المكتبى فقط .

- **SystemTables**– حدده بـ True للحصول على System Table وكذلك لكى تستخدم مجموعات البيانات من remote server . يكون هذا الـ parameter ذا قيمة لتطبيقات قاعدة بيانات الـ Server والعميل فقط .

- **List**– قم بتمرير أى خاصية TStrings أو إلى هذا الـ parameter . يقوم الـ GetTableNames بمسح القائمة وإدخال اسماء Table مجموعة بيانات فى القائمة . يستخدم التطبيق هذا الـ parameter لبدء خاصية Items للـ ComboBox2 ، وهو object كل من نوع الـ TStrings . إن تحديد الـ Item Index التابع لهذا الـ object بصفر يعرض المدخل المذكور أولاً فى نافذة تحرير صندوق الـ Combo .

توضح القائمة ١٧-٣ التركيبة الرئيسية للـ SQL Play ، والتى تؤدى أوامر الـ SQL وتعرض tables قاعدة البيانات المختارة . بعد القائمة ، سوف أوضح كيف يمكن للـ form Objects الرئيسية أن تتعاون لتعطى نتائج بحث قاعدة البيانات فى الـ DBGrid (يمكنك أن تستخدم نفس الـ methods مع أى data-aware controls .

القائمة ١٧-٣: SQLPlay\Main.pas.

```
unit Main;

interface

uses
  SysUtils, Windows, Messages, Classes, Graphics,
   Controls,
     Forms, Dialogs, StdCtrls, Buttons, ExtCtrls, Grids,
```



```
    DBGrids, DBCtrls, DB, DBTables, Open;

  type
      TMainForm = class(TForm)
        DataSource1: TDataSource;
          DBNavigator1: TDBNavigator;
          DBGrid1: TDBGrid;
          Memo1: TMemo;
          Bevel1: TBevel;
          PerformBitBtn: TBitBtn;
          CloseBitBtn: TBitBtn;
          OpenBitBtn: TBitBtn;
          Query1: TQuery;
          procedure OpenBitBtnClick(Sender: TObject);
          procedure PerformBitBtnClick(Sender: TObject);
          procedure FormClose(Sender: TObject;
           var Action: TCloseAction);
        private
       { Private declarations }
        public
       { public declarations }
       end;

  var
      MainForm: TMainForm;

  implementation

  {$R *.DFM}

  procedure TMainForm.OpenBitBtnClick(Sender: TObject);
  begin
   if OpenForm.ShowModal = mrOk then
      begin
        Query1.Close;
```



```
    try
        Query1.DatabaseName := OpenForm.ComboBox1.Text;
          Query1.SQL.Clear;
          Query1.SQL.Add('Select * From ' +
          OpenForm.ComboBox2.Text);
            Memo1.Lines := Query1.SQL;
            Query1.Open;
            Memo1.SetFocus;
            PerformBitBtn.Enabled := True;
           except;
          ShowMessage('Unable to open database');
           end;
   end;
end;

procedure TMainForm.PerformBitBtnClick(Sender: TObject);
begin
 Query1.Close;
   try
    Query1.SQL := Memo1.Lines;
      Query1.Open;
     except
    ShowMessage('Invalid query');
     end;
end;

procedure TMainForm.FormClose(Sender: TObject;
 var Action: TCloseAction);
begin
 Query1.Close;
end;

end.
```

فى قائمة البرنامج الرئيسى، يعرض الـ OpenBitBtnClick procedure نافذة OpenForm باستدعاء الـ ShowModal . إذا أنهى المستخدم الـ dialog

بضغط OK، يغلق البرنامج الـ Query1 للإعداد لفتح قاعدة بيانات جديدة. تعمل الـ Query objects مثل الـ Tables، ولكنها تستطيع أيضاً أن تنفذ عبارات SQL. وهذا يستلزم بعض عبارات الاعداد. أولاً، يقوم البرنامج بتعيين DatabaseName من نافذة تحرير الـ ComboBox1 الخاصة بالـ dialog. ثم نحن بحاجة إلى عبارتين لمسح قائمة الـ SQL string الحالى فى الـ Query، واستدعاء الـ Add لادخال عبارة مثل هذه:

Select * From Animals

إن أمر الـ SQL Select يبحث فى مجموعة بيانات مثل Animals عن مدخلات متشابهة. وتختار علامة النجمة كل أعمدة (أو حقول) الـ Animals table. ولأنه لم يتم استخدام عبارة WHERE لتحد من اختيار الصفوف، فإن كل الصفوف (أو السجلات) فى الـ Animals table قد تم ضمها إلى المجموعة الناتجة. ولكى يمكنك من تحديد أوامر الـ SQL، يقوم البرنامج بتعيين خاصية الـ SQL التابعة للـ Query1 إلى Memo. ثم يستدعى البرنامج الـ Open for Query1، والذى يصدر عبارة SQL إلى مجموعة البيانات.

Note

ملحوظة: إن استدعاء الـ Open للـ Query يصدر أمر الـ SQL Select فقط. ولأوامر الـ SQL الأخرى مثل Insert وDelete، استدع الـ ExecSQL method.

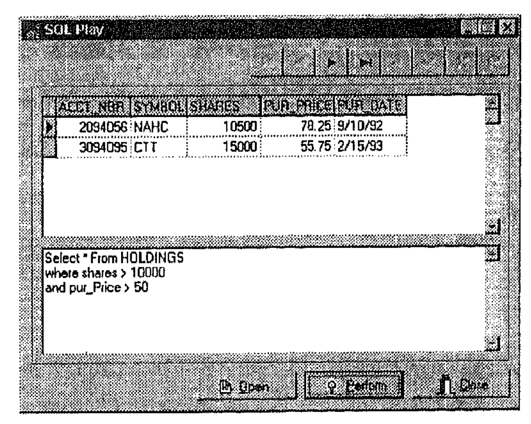
إن الـ PerformBitBtnClick procedure ينفذ أوامر SQL التى تدخلها فى نافذة Memo. أولاً، يغلق البرنامج الـ Query1. (أغلق الـ Queries والـ Tables دائماً قبل عمل أى تغييرات هامة فى خصائصهما). قم بتعيين TStrings أو TStringList إلى خاصية الـ SQL، ثم افتح قاعدة البيانات لإصدار الأمر، والذى يكون فى هذا المثال Select.

وللتجربة ببرنامج الـ SQLPlay، قم بتشغيل البرنامج وافتح قاعدة بيانات الـ DBDEMOS. اختر table مجموعة بيانات الـ HOLDINGS. اضغط OK لتعود إلى نافذة البرنامج الرئيسية، واضغط داخل الـ Memo pad. قم بتحرير عبارة الـ SQL الأصلية إلى الآتى، واضغط Alt+P لأداء الأمر، والذى يبحث فى مجموعة البيانات عن كل الممتلكات تكون فيها الأسهم أكبر من ١٠٠٠٠ :

```
Select * From HOLDINGS
where shares >10000
```

يمكنك الاستمرار فى تنقية البحث. على سبيل المثال، أضف تحديداً آخر، كما يلى، تحت السطرين السابقين لعرض الأسهم التى تزيد عن ١٠٠٠٠ ولها سعر شراء أكثر من ٥٠ دولار. يوضح شكل ١٧-٢٢ النافذة الناتجة. (إن تطبيق الـ SQLPlay على القرص المدمج يوضح كيفية إصدار والاستجابة إلى أوامر الـ SQL).

```
and pur_Price > 50
```



ACCT_NBR	SYMBOL	SHARES	PUR_PRICE	PUR_DATE
2094056	NAHC	10500	78.25	9/10/92
3094095	CTT	15000	55.75	2/15/93

شكل ١٧-٢٢: هنا، يتم توضيح بحث فى الـ Holdings DBDEMOS Table للسجلات ذات الأسهم التى تزيد عن ١٠٠٠٠، ولها سعر صرف أعلى من ٥٠ دولار.

Note

ملحوظة: بعد فتح قاعدة بيانات بواسطة الـ SQLPlay Open Button، يمكنك إختيار database Table آخر دون إعادة فتح الـ dialog. على سبيل المثال، إستبدل كل السطور فى نافذة الـ Memo بـ Select * From Animals، وإضغط Alt+P لفتح الـ Animals table وعرض كل سجلاته.

قواعد بيانات Master-Detail

إن أغلب قواعد البيانات علاقية، وهذا يعنى ببساطة إنها لها إثنين أو أكثر من الـ Tables ترتبط ببعضها بواسطة حقول عامة. على سبيل المثال، إن الـ Customer table قد يكون له حقل ID يستخدم أيضاً فى الـ Purchases tables لتعريف كل شراء يكون قد قام به عميل بعينه. وهذا الإعداد يسمى علاقة

(one-to-many) . يتم تعريف السجلات المنفردة بطريقة فريدة بواسطة مفتاح (حقل الـ ID) في table الأصل (Customer) . ويكون واحداً أو أكثر من السجلات مرتبط بالعملاء في جدول التفاصيل (Purchases) . وهذه العلاقة تسمى (master-detail model) .

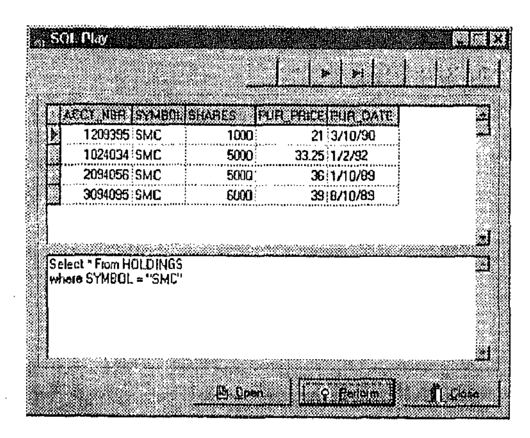
فهم (master-detail model)

يأتى Delphi بالعديد من الأمثلة المفيدة من tables قاعدة بيانات (master-detail) . وكمثال ، قم بتشغيل تطبيق الـ SQLPlay من الفصل السابق وأفتح الـ Alias name المستعار DBDEMOS أختر الـ MASTER table وإضغط OK لتعود إلى النافذة الرئيسية . يذكر الـ DBGrid object اسهم متنوعة ، كل منها معرف بطريقة فريدة بواسطة primary key يسمى SYMBOL .

الآن ، إدخل عبارة الـ SQL هذه في نافذة الـ Memo :

```
Select * From HOLDINGS
where SYMBOL = "SMC"
```

إضغط Alt+P لأداء العبارة وأختر كل الـ HOLDINGS ذات الرمز SMC . وtable الأصل له مدخل واحد فقط مثل هذا ؛ وجدول التفاصيل يمكن أن يكون له سجلات عديدة ، واحداً لكل شراء لهذا السهم بالذات . يوضح شكل (١٧-٢٣) نافذة الـ SQLPlay بعد أداء هذا البحث .



شكل (١٧-٢٣): الـ SQLPlay بعد البحث في الـ HOLDINGS على حقل الـ SYMBOL العام

برمجة تطبيقات (master-detail)

إن خـصـائص الـ MasterSource والـ MasterFields لـ Table تعـرف علاقات الـ master-detail فى تطبيقات قاعدة بيانات Delphi . أستخدم هذه الخصائص كما يلى :

- MasterSource – وكـمـا أيضـاً detail Table object ، عين اسم DataSource object مرتبط بـ master Table . أترك هذه الخاصية خالية فى الـ master Table.
- MasterFields – كذلك للـ detail Table عين string له واحداً أو أكثر من أسماء الحقل إلى هذه الخاصية ، التى تعرف الحقل (العمود) الذى يرتبط فوقه أثنين من الـ Tables.

يمكنك تحديد حقول متعددة منفصلة عن بعضها بفصلة منقوطة فى MasterFields string . ها هى بعض العينات التى توضح الـ formats الممكنة :

```
DetailTable.MasterFields := 'Symbol';
DetailTable.MasterFields := 'Symbol;Exchange';
```

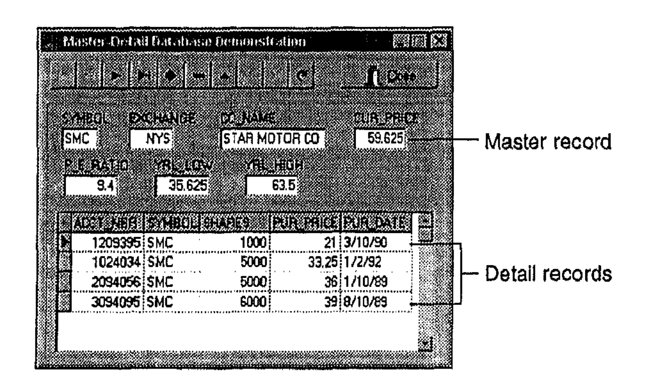
وبخلاف بعض الأراء ، لا يجب عليك أن تحدد MasterFields فى Table . قم بتعيين string لهذه الخاصية فقط فى الـ detail Table object . بالإضافة إلى ذلك ، حدد خصائص الـ IndexFieldNames أو الـ IndexName التابعة لـ detail Table . إستخدم دائماً الـ IndexName متى أمكن لأنها تساعد فى عملية lookups . حدد الـ IndexFieldNames عندما لا يوجد فهرس للـ joined field.

على القرص المدمج : كمثال على كيفية إنشاء تطبيق قاعدة بيانات الـ master والـ detail قم بتشغيل برنامج الـ Master على القرص المدمج فى دليل الـ Source\Master . يستخدم هذا البرنامج الـ Master tables والـ Holdings من قاعدة بيانات الـ DBDEMOS المتوفرة مع Delphi . يوضح شكل ١٧-٢٤ عرض البرنامج . إضغط أزرار الـ DBNavigator لتصفح الـ Master Table وعرض المدخلات من Holdings table مع حقول Symbol الملائمة . توضح الـ code التالية البرمجة الوحيدة المهمة فى هذا التطبيق : الـ



form OnCreate، الذى يفتح أثنين من الـ Tables لقاعدة البيانات. وتعتنى الـ Components وخصائصها بكل الجوانب الأخرى لعملية هذا البرنامج - وهى مظهر قوى لقدرات البرمجة البصرية لـ Delphi.

```
procedure TMainForm.FormCreate(Sender: TObject);
begin
 MasterTable.Open;
    DetailTable.Open;
end;
```



شكل (١٧-٢٤): إضغط أزرار الـ DBNavigator لترى الأسهم من Master table وتوضح أية ممتلكات لتلك الأسهم فى الـ DBGrid من الـ فشزمثdetail Holdings.

يوضح جدول ١٧-٢ قيم الخاصية الهامة للـ DataSource والـ Table التابعة لبرنامج الـ Master. بشكل عام، فإن كل علامة master-detail تحتاج إلى زوجين من هذه الـ objects المتعلقة بواسطة الخصائص الموضحة فى الـ Table. ويظهر الـ Table أيضاً data-aware control (EditSYMBOL) مرتبط بالـ MasterSource، الـ DetailDBGrid مرتبط بالـ DetailSource. وهذه البنود تمثل الحد الأدنى من العناصر المطلوبة لتطبيق قاعدة بيانات الـ master-detail. (بالمناسبة، لقد أستخدمت الـ Database Form Wizard لتبدأ هذا التطبيق، الذى قمت بتعديله بإعادة تسمية الـ components، وبإدخال DBGrid object، وبعمل تعديلات بسيطة للعرض).

إستخدام Data Modules:

ليس من غير الشائع للعديد من التطبيقات أن تتوصل إلى نفس database tables. على سبيل المثال، بمصاحبة قاعدة بيانات مطولة، فقد يطلب منك أن تنشئ عشرات من التطبيقات لبحث، وتسهيل، وتصنيف معلومات قاعدة البيانات. فى تلك الحالات، من المستحسن عادة أن تنشئ Data Module (أو Data Module) تعرف القواعد واللوائح للتوصل إلى database tables. بتركيب الـ Data Module فى الـ Object Repository الخاص بـ Delphi، فيمكنك أنت والمبرمجون الآخرون أن تبدأوا بسهولة تطبيق قاعدة بيانات جديد.

جدول (١٧-٢): خصائص الـ DataSource والـ Table لتطبيق الـ Master.

Component	*Name*	*Property*	*Value*
Table	MasterTable	DatabaseName	DBDEMOS
		TableName	master.dbf
DataSource	MasterSource	DataSet	MasterTable
Table	DetailTable	DatabaseName	DBDEMOS
		IndexName	SYMBOL
		MasterFields	SYMBOL
		MasterSource	MasterSource
		TableName	holdings.dbf
DataSource	DetailSource	DataSet	DetailTable
DBEdit	EditSymbol	DataField	SYMBOL
		DataSource	MasterSource
DBGrid	DetailDBGrid	DataSource	DetailSource

تحتوى الـ Data Module النموذجية على الـ Table والـ Query والـ DataSource و components أخرى والتى تعرف بـ Alias name، اسم الـ Table وخصائص أخرى. يمكن لأى عدد من التطبيقات أن يضيف الـ Data Module ملفات المشروعات الخاصة بهم، ويبدأوا فى إستخدام Tables قاعدة البيانات على الفور. وهذا التنظيم أيضاً حافظ على components قاعدة البيانات الغير بصرية فى نافذة منفصلة غير مرئية، بدلاً من التكدس فى form وقت

التصميم . ولكن الميزة الحقيقية لإستخدام الـ Data Modules هى توفير تطبيقات متعددة مع المواد الهام اللازمة للتوصل إلى قاعدة اليانات . بالإضافة إلى ذلك ، إذا تغيرت القواعد واللوائح - كما هو الحال غالباً - يمكنك ببساطة تحديث الـ Data Modules وإعادة الـ compile لتطبيقاتك بدلاً من إعادة برمجة الـ DataSource والـ Table فى كل برامجك .

Note

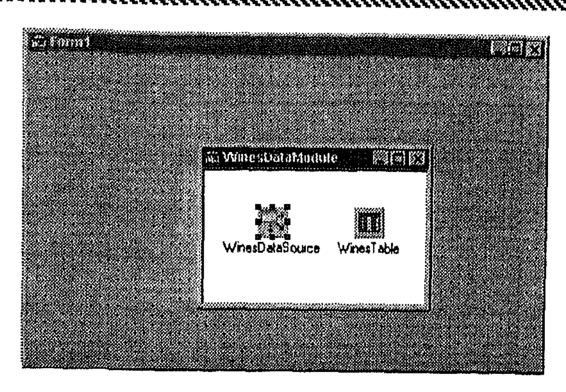
ملحوظة: كمقدمة حول إستخدام الـ Data Modules والـ Object Repository، أنظر الباب الثالث، «معلومات حول الـ forms»، تحت عنوان Form Templates.

إتبع هذه الخطوات لإنشاء Data Module لقاعدة بيانات الـ Wines فى هذا الباب، ولتركيبها فى الـ Object Repository الخاص بـ Delphi (الملفات الناتجة ليست على القرص المدمج - فيجب أن تنشئها لكى تستخدم المعلومات الواردة فى هذا الفصل) :

١ - قم بتوفير دليلاً لتخزين الملفات . أبدأ تطبيقاً جديداً ، ثم إختر أمر الـ File|New Data Module .

٢ - إستخدم الـ Object Inspector ليصبح WinesDataModule . يمكنك حفظ كل الملفات عند هذه النقطة لإقامة الدليل المنشود . أعد تسمية الملف المرتبط بالـ Data Module - (وهو Unit2.pas فى هذا المثال) ليصبح شيئاً مناسباً مثل WinesDM.pas .

٣- قم بإظهار نافذة الـ WinesDataModule مرة أخرى (إستخدام أمر الـ View|Window List إذا لم تجدها) . فى هذه النافذة، قم بإسقاط DataSource وTable من الـ Data Access . قم بتغيير خاصية الـ DataSource1 Name لتصبح WinesTableSource . قم بتغيير خاصية الـ Table 1 Name لتصبح WinesTable . يوضح شكل ١٧-٢٥ نافذة الـ Data Module فى هذه المرحلة . تكون نافذة الـ Form1 خالية ولن تستخدم (ولكن، يمكنك تطويرها لتكون برنامج إختيارى للـ Data Module هذه) .



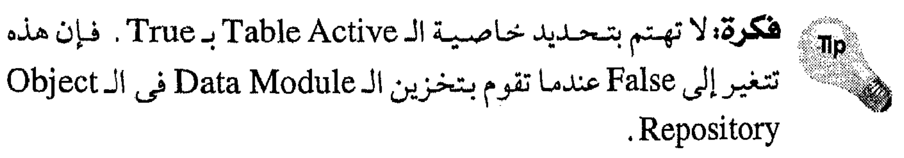
شكل (١٧-٢٥): نافذة الـ WinesDataModule ذات الـ DataSource والـ Table.

٤- بعـد ذلك تقـوم بتطوير القـواعـد الخـاصـة بالتوصل إلى قـاعـدة البيـانات بإستخدام الـ WinesTable ، وفى الـ Object Inspector ، إضـغط السـهم المشير إلى أسـفل الـواقع بعـد خـاصيـة الـ DatabaseName ، وإختـر WINES . بنفس الطريقة ، إختر Wines.db لخاصية الـ TableName .

٥- إختر الـ WinesDataSource فى الـ Data Module ثم ، بإستخدام الـ Object Inspectory ، حدد خاصية الـ DataSet بـ WinesTable .

٦- إحفظ كل الملفات بإختيار File|Save All أو بضغط على Save All Button .

تتم الخطوات السـابقة البرمـجة للـ Data Module . فى تطبيـقـاتك ، تكون حـراً فى إضـافـة components إضـافـيـة وتحـديد خـصـائص أخـرى للوصـول إلى Tables قاعدة البيانات الخاصة بك . بالرغم من أن المثال تستخدم إثنين فقط من الـ components ، فقـد يكون الـ Data Module الأكثـر تعقـيداً يملك أكثـر Table مختلفه و DataSource و component قاعدة بيانات أخرى عديدة للوصول إلى العديد من الـ Tables .

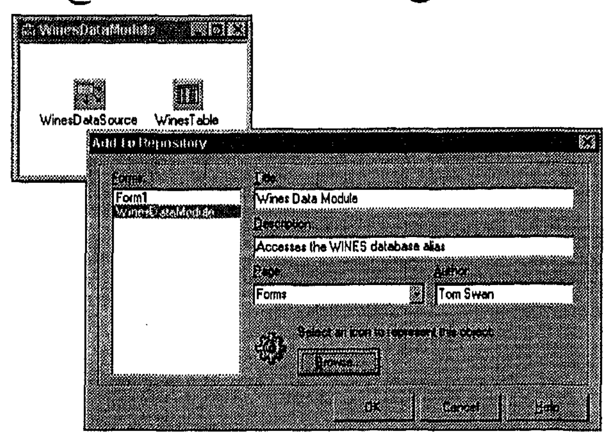


فكرة: لا تهـتم بتـحـديد خـاصـيـة الـ Table Active بـ True . فـإن هذه تتغير إلى False عندما تقوم بتخزين الـ Data Module فى الـ Object Repository .

لكى تجعل الـ Data Module متاحة لتطبيقات أخرى ، قم بإضافتها إلى الـ Object Repository الخاص بـ Delphi (سوف أشرح كيفية حذف المدخل ، فلا

تتـردد فى أن تجـرب هذا) . أتبع هذه الخطوات لإضـافـة الـ Data Module من الخطوات السابقة إلى الـ Object Repository .

١ - إجـعل نافـذة الـ WinesDataModule مـرئيـة (إسـتـخـدم الـ View|Window List إذا لزم الأمر لتحديد موضع النافذة) . إضـغط يميناً أثناء الإشارة بمؤشر الفأرة داخل النافذة . هذا يؤدى إلى إظهار قائمة أختر أمر الـ ... Add to Repository . يوضح شكل ١٧-٢٦ الـ dialog الناتج .



شكل (١٧-٢٦): بالإضافة إلى الـ Add to Repository dialog.

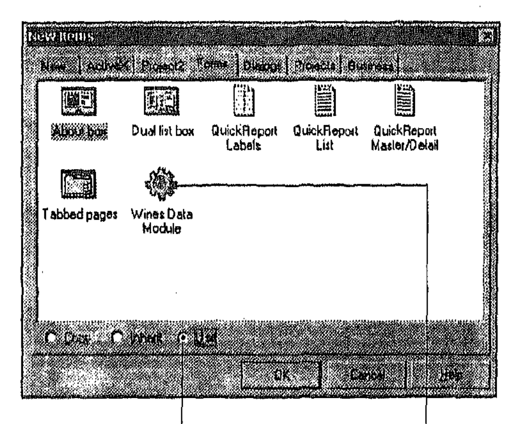
٢ - أختـر الـ WinesDataModule فى قـائمـة الـ Forms (يجب أن يتم إختيارها حسب النظام الإفتراضى) . قم بإدخال Title و Description، وأختر Page حيث تريد أن تظهر الـ Data Module عندما تختار الـ ....File|New . يمكنك أيضاً أن تملأ حقل الـ Author وتضغط Browse لإختيار ملف أيقونة لعرض صورة فى الـ New Items dialog . (لقد أستخدمت أيقونة مقدمة مع Delphi فى دليل الـ Images\Icons) . يوضح شكل ١٧-٢٦ القيم التى ملأتها .

٣ - إضغط Ok لإغلاق الـ Add to Repository dialog إذا لم تكن قد حفظت الـ module المضافة ، فإنه سيطلب منك أن تفعل هذا قبل تضمينها فى مخزن Delphi .

يمكنك الآن إستخدام الـ Data Module فى أى تطبيق هذا أمر بسيط - فقط أخترها بإستخدام أمر الـ File|New . حاول تجربة هذه الخطوات لتختبر الـ Wines Data Module :

١ - إبدأ تطبيقاً جديداً.

٢- أختـر ...File|New وإضـغط Forms page tab فى الـ New Items dialog. يجب أن ترى الـ Wines Data Module التى تم إضـافتها حديثاً. (أنظر شكل ١٧-٢٧).



شكل (١٧-٢٧): أختر الـ Wines Data Module التى تم إضافتها حديثاً من الـ New Items dialog.

٣- إضـغط الزر الذى يحمل بطاقة Use قرب أسفل نافذة الـ dialog (راجع شكل ١٧-٢٧). أنظر الملحوظة الموجودة بعد هذه الخطوات للحصول على تفسير للطرق الثلاثة الممكنة لتضمين الـ module : Copy، Inherit، وUse.

٤- إضغط مرتين أيقونة الـ Wines Data Module . هذا يفتح نافذة الـ Data Module فى التطبيق الحالى وأضف source Pascal code unit الخاصة به إلى نافذة الـ code.

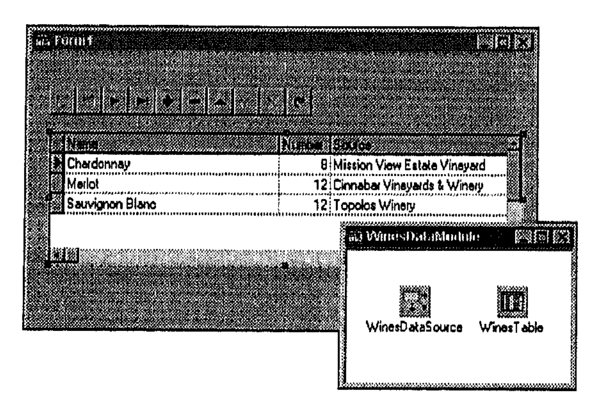
٥- إنتـقل إلى نافـذة الـ code، وأخـتـر Unit1 (الـ form source code الرئيسية للبرنامج). ضع تعريف الـ uses فى أعلى الشاشة وقم بتعديله كما يلى. هذا التغيير يجعل الـ Data Module، WinesDM (هذا هو الأسم الذى حفظت به الـ source Pascal code للـ module، متـاحـة للـ form unit الرئـيسـيـة. بالتبادل، يمكنك إستخدام أمر الـ File|Use Unit الخاص بـ Delphi لتقوم بهذا التغيير بصورة تلقائية، ولكن لقطاع الـ implementation الخاص بالـ unit المضيفة بدلاً من واجهة التطبيق كما هو موضح هنا:

```
uses
    Windows, Messages, SysUtils, Classes, Graphics,
   Controls,
    Forms, Dialogs, WinesDM;
```

٦- أضف DBNavigator وDBGrid من الـ Data Controls palette إلى نافذة الـ form الرئيسية. أختر كلا الـ objects (clock+Shift لكل object). في نافذة الـ Object Inspector، حدد خاصية الـ DataSource، حدد خاصية الـ DataSource لكلا الـ object بما يلي (هذا هو الإختيار الوحيد في هذا المثال):

WinesDataModule.WinesDataSource

٧- إرجع إلى نافذة الـ Wines Data Module إذا لزم الأمر). أختر الـ WinesTable، وإستخدم الـ Object Inspector لتحديد خاصية الـ Active بـ True. هذا يفتح قاعدة بيانات الـ Wines ويظهر البيانات الحية في نوافذ الـ DBNavigator والـ DBGrid (أنظر شكل ١٧-٢٨).



شكل (١٧-٢٨): هذا البرنامج، الذى تم إنشاؤه بإتباع الخطوات فى هذا الفصل، يستخدم الـ Wines Data Module للوصول إلى قاعدة Wines Data Module للوصول إلى قاعدة بيانات الـ Wines.

إن التطبيق الذى أنشأته لتوك بإستخدام الـ Wines Data Module يوضح طريقة عظيمة للتشارك فى objects الوصول إلى قاعدة البيانات مع تطبيقات متعددة إنك لا تحتاج إلى DataSource فى الـ form الرئيسية للبرنامج لأن هذه متوفرة فى الـ Data Module. تستطيع التطبيقات الأخرى أن تستخدم نفس الـ

Data Module للتمكن من الوصول إلى قاعدة بيانات الـ Wines . إذا إحتاجت هذه الـ objects إلى التغيير - مثلاً، لتغيير الـ alias name ، اسم الـ Table، أو لإضافة حقول وهمية كما هو موضح فى هذا الباب بإستخدام الـ phanton Fields - فإنك ببساطة تقوم بإعادة الـ compile لتطبيقاتك لتضمين التغييرات .

Note

ملحـــوظة: إن البنود الموجـــودة فى الـ Object Repository يمكن إضافتها إلى مشروع بإستخدام ثلاثة خيارات : Copy، Internet، وUse (لاحظ أن ليس كل الخيارات متاحة لكل نوع من البنود - المشروعات، مثلاً، يمكن نسخها فقط) . عندما تقوم بنسخ عنصر Object Repository، يصنع Delphi نسخة كاملة لملفات ذلك العنصر فى دليل مشروعاتك . وأية تغييرات تفعلها بهذا العنصر توجد فى هذه النسخة فقط . عندما تورث بنداً، ينشئ Delphi الـ class الجديدة على أساس item's class مع هذا الخيار، يمكنك فعل تغييرات، ولكن بواسطة إعادة الـ compile، قم بتوريث أية تغييرات صنعت بالعنصر فى الـ Object Repository . عندما تستخدم بنداً، يشير Delphi إلى ملفاته أينما كانت مخزنة . أية تغييرات تحدث بالعنصر تنعكس فى تطبيقك عندما يتم الـ compile . إن قواعد بيانات الـ Data Modules أفضل ما تكون عند توريثها أو إستخدامها .

برمجة قاعدة البيانات

تصف الفصول التالية بعض البنود المتنوعة حول برمجة قاعدة البيانات مع Delphi والتى قد تجدها معاونة لك .

خاصية الـ TDataSet CacheBlobs

إن الـ TDataSet class توفر الآن خاصية، وهى الـ CacheBlobs، يمكنك أن تحددها بـ True أو False إن الـ CacheBlobs يتم توارثها من جانب كل الـ class المنحدرة من جانب الـ class المنحدرة من الـ TDataSet، مثل الـ TBDEDataSet، TTable، TQuery، و TStoredProc .

حدد الـ CacheBlobs بـ True لتحفظ فى الذاكرة أى عدد من حقول الـ Blob بقدر الإمكان، حتى عندما تكون السجلات المرتبطة بها ليست فى الذاكرة أى

عدد من حقول الـ Blob بقدر الإمكان، حتى عندما تكون السجلات المرتبطة بها ليست فى الذاكرة.

حدد الـ CacheBlobs بـ False لإخراج حقول الـ Blob من الذاكرة مع السجلات الأخرى لقاعدة البيانات.

والطبيعى هو أن تكون الـ CacheBlobs محددة بـ True. هذا يؤدى إلى تحسين الأداء خاصية فى البرنامج التى تعرض Bitmap من حقول الـ Blob. عندما يرجع المستخدم إلى سجل ذا CacheBlobs، فيتم تحميله من الذاكرة بدلاً من Table قاعدة البيانات على قرص أو gride محددة بـ True.

الـ TDBDataSet مقابل الـ TDataSet

فى Delphi 4، لم تعد الـ TDataSet class معتمدة على الـ BDE. وهذا التغيير ليس له أثر على التطبيقات الموجودة، ولا تؤثر على إستخدام الـ TDBDataSet (التى تنحدر منها الـ TTable، الـ TQuery، والـ TStoredProc). ولكن، لقد أصبح ممكناً الآن أن تنشئ TDataSet classes لا تحتاج إلى الـ BDE للتوصل إلى قواعد البيانات. يمكنك، مثلاً، أن تنشئ الـ TDataSet class الخاصة بك للوصول إلى Server قاعدة بيانات دون الخوض فى الـ BDE - لا يجب حتى تركيب الـ BDE على نظام المستخدم النهائى.

يعتبر إنشاء TDataSet class أمراً معقداً ويتطلب مهارات برمجة متطورة. إن class معنوية، بمعنى أنه لا يمكن تمثيلها كـ object. لإستخدام الـ TDataSet class الجديدة، يجب أن تستنبط class جديدة وقم بتوفير الـ code للـ methods المعنوية. كيف تفعل هذا؟ إن هذا يعتمد بدقة على Server قاعدة البيانات الخاص بك، وهو خارج نطاق هذا الكتاب.

إذا كنت تريد أن تنشئ الـ class الخاصة بك المشتقة بالتعامل مع الـ TBDE Dataset والـ TDB Dataset من الـ TDataSet، يمكنك أن تبدأ بدراسة كيف تقوم الـ Inprise Corporation بتطبيق الـ TBDE Dataset والـ TDB Dataset ومن الضرورى أنك تحتاج أن تنشئ classes مشابهة تستدعى النظم الفرعية لبرنامج تشغيل قاعدة البيانات الخاص بك بدلاً من إستخدام الـ BDE API. بعد إتمام هذه الخطوات، يمكنك إنشاء class شبيهة بالـ TTable class وتستخدمها فى

تطبيـقـاتك . إن كل الـ data-aware controls والـ DataSource تعـمل مع الـ TDataSet class المشتقة الخاصة بك .

وهذا يشمل الـ TClientDataSet ، والتى يسهل تنفيذها إلى حد كبير عن الـ TBDEDataSet :

خاصية الـ TField. IsBlob:

إن كل الحقول فى جداول مجموعة البيانات يتم تمثيلها فى الـ Delphi كـ objects من الـ TField class والـ classes ذات الأنواع الخـاصـة مـثل الـ TStringField والـ TIntegerField تنحدر من الـ TField class .

والـ IsBlob ، ترجع بـ True إذا كـان الـ TField object من نوع Blob (تذكر أن الـ Blob هو Binary Large Object) . فيتم تعريف الـ function كما يلى :

```
class function IsBlob: Boolean; override;
```

فى الـ procedure يعـرف الـ TField parameter ، يمكنك إسـتـدعـاء الـ IsBlob لتـحـدد مـا إذا كـان الحقل Blob أم لا وهذا قـد يسـهل عـمليـة عـرض الـ bitmap مقابل عرض النص وقيم أخرى ، عى سبيل المثال ، بعد إستخدام محرر الـ Fields ، يمكنك إخـتـيـار الـ TField من الـ Object Inspector وتنشىء event handler لـ event مثل الـ OnGetText . (لفتح محرر الـ Fields ، إضغط مرتين Table object ، إضغط يميناً داخل نافذة محرر الـ Fields ، وإستـخدم أوامر الـ Add Fields والـ New Field كما هو موضح فى هذا الباب) .

يستطيع الـ event handler أن يتأكد ما إذا كان الحقل Blob ويتخذ العمل المناسب .

يمكنك برمجة الـ event handler مع السطور التالية (هذا يفترض أنك قد كتبت الـ DisplayAsBlob procedure لعرض الـ TBlobField كـ bitmap ربما ربما بمساعدة الـ Image object فى الـ form :

```
procedure TMainForm.Table1YourFieldNameGetText(Sender: TField;
 var Text: String; DisplayText: Boolean);
begin
```

```
if Sender.IsBlob then
 begin
  DisplayAsBlob(Sender as TBlobField);
     Text := '';
      else
    Text := Sender.AsString;
end;
```

أفكار للمستخدم الخبير

- فكر فى إستخدام الـ DDB للتشارك فى البيانات فيما بين تطبيقات قاعدة بيانات متعددة إن هذا يعتبر أكثر فعالية (وبالتأكيد هو مألوف أكثر للمستخدم) من كتابة بيانات للملفات القرص المؤقتة، والذى تعد عمل عادى. أنظر الباب السادس عشر لمعرفة تعليمات حول إقامة محادثات الـ DDE.

- كما ذكرنا عدة مرات فى هذا الباب، قم دائماً بتسجيل Alias name بإستخدام الـ Database Desktop، ولا تكتب فى الـ code أى معلومات عن اسم المسار وبرنامج تشغيل قاعدة البيانات فى تطبيقاتك. بالرغم من أن هذا يسهل فعله، إلا أن أسماء المسارات قد تعيقك عند تحريك ملفات قواعد البيانات إلى مواضع جديدة، أو محركات صلبة أخرى أو إلى الشبكات. إذا قمت بإستخدام aliases للوصول إلى جداول قاعدة البيانات الخاصة بك، فإنك لن تفقد أثر مكانها، يجب أن يتم نقلها إلى محرك صلب أكبر وأحدث.

- ينشئ الـ Database Form Wizard الـ form الثانوية لتطبيق جديد. لإستخدام هذه الـ form كنافذة رئيسية للبرنامج، إضغط زر الـ Option التابع للـ Project Manager، وخصص الـ form تحت الـ Main form. يمكنك عندئذ إستخدام الـ Project Manage ليحذف الـ form الحالية للمشروع.

- إن برنامج تشغيل الـ Windows ODBC للـ Access قد تم تصميمه للإستخدام مع الـ Microsoft Office. لإنشاء تطبيقات قاعدة بيانات Access مع Delphi، يجب أن تحصل على الـ ODBC Desktop Driver التام.

إن برنامج تشغيل متوفر مع الـ Microsoft وعلى الـ ODBC SDK، والتى تشــمل برامج تشــغــيل لقــواعــد البــيــانات Access ، Btrieve، dBASE، Excel، Fox، Paradox، Text .

● لتوزيع تطبيق قاعدة البيانات ، إنك تحتاج أن تزود ملف الـ exe . التابع لبرنامجك (بالإضافة إلى أى ملفات مؤيدة للبيانات) والـ Inprise Corporation Database Engine . أنظر ملف الـ Redist.txt المتــوفــر مع Delphi لمعــرفــة معلومات حول الملفات اللازمة لتزويد المستخدمين النهائيين لك .

المشروعات التى يمكنك تجربتها

١٧-١ : أكتب تطبيق لـ Table يحتوى على الأسم والعنوان . أو فكر فى أى تطبيق آخر بسيط إستخدم الـ Database Form Wizard لإنشاء محرر ومتصفح مناسبين .

١٧-٢ : قم بتوسيع قاعدة بيانات الأسم والعنوان الخاصة بك لتشمل رقم تعريف فريد لكل سجل . قم ببرمجة Tables إضافية وتطوير علاقات master-detail . على سبيل المثال ، بدلاً من تخزين البيانات الهامة مثل أعياد الميلاد والذكرى السنوية مباشرة فى سجلات الـ master ، قم بإنشاء Dates detail table مرتبط بأسماء الأصل . مع هذا الإعداد ، يمكن لأى اسم أن يكون له عدد غير محدود من التواريخ الهامة . وإنك توفر المساحة على القرص أيضاً ، بعدم حجز مساحة فى السجلات لا تريدها لتسجيل تواريخ .

١٧-٣ : إستخدم تطبيق الـ SQLPlay الخاص بهذا الباب لإكتشاف dataset أخرى فى قاعدة البيانات DBDEMOS المتوفرة مع Delphi . إختبر مع أوامر الـ SQL Select للبحث عن مسجلات ذات قيم حقول محددة .

١٧-٤ : قم بتجربة خيارات Database Form Wizard أخرى . على سبيل المثال ، يمكنك إستخدام الـ Export لإنشاء علاقات الـ master-detail ، ولإدخال Query objects لتنفيذ البحث .

الملخص :

• تنقسم components قاعدة البيانات الخاصة بـ Delphi إلى مجموعتين : الـ Data Access التى توفر gateways و links بين التطبيق وقواعد البيانات ؛ و Data Controls والتى تشمل data-aware controls والـ objects الأخرى لعرض وتحرير حقول البيانات ، وللملاحة فى سجلات قاعدة البيانات .

• إن نسخ الـ Desktop ، Professional ، Client-Server لـ Delphi تشمل الـ Borland Database Engine (BDE) ، الذى يوفر مجموعة كاملة من برمجة الـ API functions للعديد من نظم قواعد البيانات الشهيرة مثل الـ dBASE مثل الـ Microsoft Access و Btrieve . تشمل نسخة الـ Client-Server من Delphi كل مميزات نسخة الـ Desktop بالإضافة إلى أنها توفر الوصول إلى Server قاعدة البيانات مثل الـ Oracle ، Sybase ، Microsoft SQL Server ، Interbase ، و Informix .

• إن الـ TDataSet الجديد لم يعد يعتمد على الـ BDE . يستطيع المبرمجين المتقدمين الآن أن يشتقوا class جديدة من الـ TDataSet لإستدعاء وظائف فى Server قاعدة بيانات غريبة دون تركيب أو إستخدام الـ BDE . وهذا غير ضرورى لأغلب التطبيقات .

• كبداية لإنشاء تطبيق قاعدة بيانات جديد ، إستخدم أمر الـ Database Form Wizard الموجود فى قائمة الـ Database الخاصة بـ Delphi (لقد كان يلقب هذا الأمر بـ Database Form Expert وكان موجود فى قائمة الـ Help) . ينشئ الـ Database Form Wizard الـ form التى تملك Data Access و Data Controls والتى يمكنك تعديلها بعد توليد الـ form .

• الـ Table يعادل dataset واحدة ، وهى تجمع لخلايا البيانات الذى يشكل صفاً وعموداً . إن الصفوف تعادل السجلات ؛ والأعمدة تساوى الحقول . تعتبر قاعدة البيانات مجموعة من table واحد أو أكثر . والـ Alias name هو اسم يخفى معلومات المحرك والشبكة والمسار واسم الملف للملفات وأدلة قاعدة البيانات الفعلية إستخدم دائماً والـ Alias name للإشارة إلى قواعد البيانات ؛ بهذه الطريقة تعمل تطبيقاتك بدون إعادة الـ compile إذا قمت بتحريك ملفات قاعدة البيانات إلى أدلة أخرى ، أو إذا قمت بتركيبها على شبكة بعد أن تختبر الـ code .

● تحتاج غالبية تطبيقات قواعد البيانات إلى Table وDataSource. يوفر الـ Table بوابة إلى Table قـاعـدة البـيـانات (الذى يفـضل الإشـارة إليـه بـ Alias name). يربط الـ DataSource الـ Table بواحـــد أو أكـــثـــر من الـ Data Controls.

● إستخدم الـ Database Desktop لإنشاء وتعديل Table قاعدة البيانات. يمكنك أيضاً رؤية وتحرير معلومات قاعدة البيانات بإستخدام هذا البرنامج التوضيحى، والذى هو بمثابة نهاية أمامية للـ BDE.

● توفر الـ BDE مجموعة فرعية من أوامر الـ SQL المعيارية (وهى Select، Insert، Update وDelete) لـ formats قاعدة البيانات Paradox، dBASE، وOracle وformat قواعد البيانات الأخرى.

يمكنك إستخدام اللغة الكاملة مع Servers آخرين إعتماداً على تأييد الـ SQL الخاصة بهم.

● ينفذ الـ Query عبارات الـ SQL. لأن components قاعدة بيانات Delphi توفر الكثير مما تستطيع الـ SQL القيام به، فإنك غالباً تحتاج إلى إستخدام أمر الـ SQL Select فقط لأداء البحث، مثلاً، ولتقصر الـ dataset على السجلات التى تلائم المتغييرات المتنوعة.

● إن تطبيق قاعدة البيانات الـ master-detail يربط إثنين من Tables قاعدة البيانات بإستخدام حقل مشترك مثل رقم العميل أو الرمز التجارى لسوق البورصة.

إن تركيبة master-detail تتطلب زوجين من الـ Table والـ DataSource المتعلقة ببعضها.

● فكر فى وضع الـ الـ Table والـ DataSource والـ Data Access الأخرى الخاصة بك فى Data Module، ثم قم بتركيبها فى الـ Object Repository الخاص بـ Delphi. تستطيع التطبيقات المتعددة عندئذ أن ترث أو تستخدم الـ Data Module لتتمكن من الوصول إلى Tables قاعدة البيانات.

يحتوى الباب التالى على قصة برمجة قاعدة البيانات. سوف تتعرف على كيفية تطوير التقارير والـ Charts بإستخدام معلومات قاعدة البيانات.

الباب الثامن عشر
تطوير الـ Charts والتقارير

محتويات هذا الباب:

- Components
- إنشاء الـ charts
- البداية مع الـ charts
- طباعة واصدار الـ charts
- Chart data sources
- إنشاء التقارير
- البداية مع التقارير
- طباعة عناوين العمود
- طباعة معلومات النظام
- تلخيص الاعمدة
- ترتيب بيانات التقرير
- طباعة التقارير فى وقت التشغيل

يستطيع أى شخص ان يخزن بيانات فى الحاسوب ، ولكن استخلاص معلومات من قاعدة بيانات وتقديمها فى شكل قابل للاستخدام هو ما يتطلب مهارة حقيقية . ان رؤية البيانات هو ما يجعل المعلومات مفيدة وهامة ان واضعى برامج قاعدة البيانات ومديرو الانظمة قد امضوا غالباً كثير من الوقت فى تطوير التطبيقات لرؤية وطباعة الـ charts والتقارير عن أى نوع آخر من البرمجيات .

ولقد جعل Delphi هذه المهام سهلة كما ستعرف فى هذا الباب ، ان مكتبة الـ TeeChart 4 الخاصة بـ Delphi ، التى أنشأها ديفيد برنيدا ، توفر العديد العديد من انواع الـ charts لعرض البيانات بصرياً . ومكتبة الـ QuickReport 3 الخاصة بـ Delphi، التى أنشأها آلان لوكيرت ، توفر مكتبة components متنوعة لتطوير كل انواع التقارير المطبوعة . يوفر Delphi 4 نسخ كاملة معيارية لكل مكتبة ، مع components مركبة على الـ VCL palette ، جاهزة للاستخدام .

فى هذا الباب ، سوف اغطى تقنيات البرمجة المتوسطة والاساسية لمكتبات الـ TeeChart والـ QuickReport لإنشاء knock-your-socks-off printouts محترفة من معلومات مخزنة فى جداول قاعدة البيانات .

يأتى Delphi 4 مزوداً بطبعات معيارية لمكتبات الـ TeeChart والـ QuickReport، والتى يتم تغطيتها فى هذا الباب .

Components:

ان Delphi التى يتم تغطيتها فى هذا الباب هى :

- **Chart:** استخدم هذا الـcomponents لإنشاء charts من بيانات مخزنة فى ملفات ، أو دخلت مباشرة فى برنامج . الـ Palette : Data Controls .
- **DBChart:** استخدم هذا الـcomponents لإنشاء charts من بيانات مخزنة فى Tables قاعدة البيانات . الـ Palette : Data Controls .
- **DecisionGraph:** استخدم هذا الـcomponents لإنشاء الرسوم البيانية من بيانات قاعدة البيانات . الـ Palette : Decision Cube .
- **QRBand:** تتكون التقارير من bands ، كل واحدة منها تعتبر object من هذا الـcomponents . الـPalette : QReport .
- **QRChart:** استخدم هذا الـcomponents لإدخال charts فى تقارير قاعدة البيانات . الـ Palette : QReport .
- **QRChildBand:** هذا الـcomponents يتم استخدامه لإنشاء bands ممتدة مع components اخرى تتحرك أو تمتد هى الاخرى ، وللـ bands التى تمتد فوق صفحات متعددة . الـ Palette : QReport .

- **QRDBImage**: استخدم هذا الـcomponents لإدخال Image فى تقارير ، غالباً يكون واحداً مرتبطاً بحقل Blob فى Table قاعدة البيانات . الـ QReport: Palette .

- **QRDBRichText**: استخدم هذا الـcomponents لإدخال Rich text object فى التقرير. الـ QReport: Palette .

- **QRDBText**: استخدم هذا الـcomponents اغلب انواع حقول Table قاعدة البيانات فى هيئة نص. وهذا ينطبق ليس فقط على حقول النص، ولكن ايضاً على الحقول الرقمية، حقول البيانات، حقول العملة، والحقول الاخرى. الـ QReport: Palette .

- **QRExpr**: استخدم هذا الـcomponents لإنشاء تعبيرات مثلاً، لحساب قيمة مدخل التقرير باستخدام حقول Table قاعدة البيانات اخرى. الـ QReport: Palette .

- **QRImage**: استخدم هذا الـcomponents لإدخال bitmap أو نوع آخر من صور مثل الايقونة فى التقرير. يؤيد هذا الـcomponents كل انواع الصور مثل الـImage . الـQReport: Palette .

- **QRLabel**: استخدم هذا الـcomponents لإدخال label ثابتةفى تقرير الـQReport: Palette .

- **QRMemo**: استخدم هذا الـcomponents لإدخال objects نص متعدد الاسطر فى تقرير. الـQReport: Palette .

- **QRPreview**: استخدم هذا الـcomponents لتوفر مراجعة على الشكل النهائى للتقرير ولكن بدلاً من استخدام هذا الـcomponents، من السهل ان تستدعى الـ Preview method. للـQuickRep . الـQReport: Palette .

- **QRRichText**: استخدم هذا الـcomponents لإدخال rich text object فى تقرير. الـQReport: Palette .

- **QRShape**: استخدم هذا الـcomponents لإدخال objects جرافيك مثل المستطيل أو الدائرة فى تقرير . الـ Palette : QReport .

- **QRSubDetail**: استخدم هذا الـcomponents لربط dataset المتعددة فى تقرير . إنك تفعل هذا لطباعة تقارير عن قواعد البيانات التى تستخدم تركيبة الـ master-detail . الـPalette : QReport .

- **QRSysData**: اســتــخـــدم هذا الـcomponents لإدخـــال system objects متنوعة مثل الوقت والتاريخ الحالى، ارقام الصفحات، والعناصر المتنوعة الاخرى فى تقرير، غالباً فىheader band . الـ Palette : QReport .

- **QuickRep**: استخدم هذا الـcomponents لإنشاء تقرير جديد تقوم خاصية الـ bitmas للcomponents بادخال انواع متعددة من الـ bitmas، والتى تحمل حقول قاعدة البيانات والانواع الاخرى من الـ objects التى سيتم طباعتها فى التقرير التام . الـPalette : QReport .

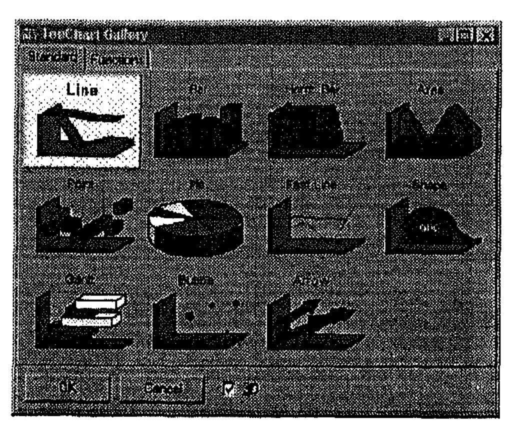
Note

ملحوظة: ان كثير من الـ QR (QuickReport) يتم إنشاءها من Data Control . اما الـ components الاخرى يتم إنشاءها من controls غير controls قواعد البيانات . على سبيل المثال، يمكن للـQRImage ان يدخل bitmap فى تقرير؛ ويفعل الـQRDBImage نفس الشئ، ولكنه يتلقى بياناته من حقل Table قاعدة بيانات (فى هذه الحالة، يكون غالباً حقل Blob، أو الـ Large Object, field).

إنشاء الـCharts مع الـTeeChart:

ان كل نوع من الانواع العديدة المختلفة من الـ charts فى الـ TeeChart library يتم إنشاؤه من سلسلة من نقاط البيانات . وكل سلسلة هى نفسها object ينتمى للـchart . يوضح شكل (١٨-١) الـ TeeChart Gallery dialog الذى تستطيع ان تختار منه نوع السلسلة التى تريد استخدامها فى الـ charts وكل سلسلة chart لها خيارات متعددة لتخصيص النتائج كما تشاءها . يمكنك إضافة labels، تعديل الوان، اختيار حدود متنوعة، تحديد تأثيرات الـ3D، والاختيار من بين خيارات اخرى عديدة.

ملحوظة: ان التوثيق الكامل للـ TeeChart library متوفر فى دليل تركيب Delphi . (لأننى استخدم نسخة سابقة على Delphi 4، فلا اعلم فى هذه اللحظة اين سيكون موضع ملف التوثيق، ولكن فى Delphi 3، كان فى مسار دليل الـ Delphi 3\TeeChart). افتح ملف الـ TChartVx.doc (x تساوى رقم النسخة) باستخدام متصفح الـ Microsoft Word أو الـ Word و online help الكاملة متاحة ايضاً- اختر أى TeeChart أو بنداً آخر فى نافذة Delphi form واضغط F1 لمزيد من المعلومات.



شكل (١٨-١): لاختيار نوع الـ charts Series التى تريدها، اختر ايقونة من الـ TeeChart Gallery dialog. يمكن ان تعرض كل chart فى from ثنائية أو ثلاثة الابعاد

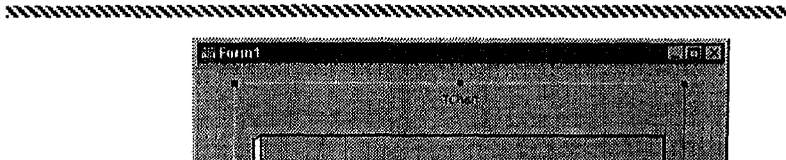
البداية مع الـ charts:

ان اسهل الطرق للبداية فى استخدام الـ TeeChart library هو إنشاء charts بسيطة. إبدأ Delphi ثم إتبع هذه الخطوات (لايجب عليك ان تحفظ المشروع):

١ - إبدأ تطبيقاً جديداً.

٢- اضغط الـ Additional page tab على الـ VCL palette لـ Delphi.

٣- اضغط ايقونة الـ Chart، ثم إضغط داخل نافذة الـform لوضع Chart object جديد خالى. تبدو نافذة الـform على شاشتك مثل شكل (١٨-٢).



شكل (١٨-٢): استخدام الـ Chart لإنشاء Chart جديد خالى فى نافذة الـ form

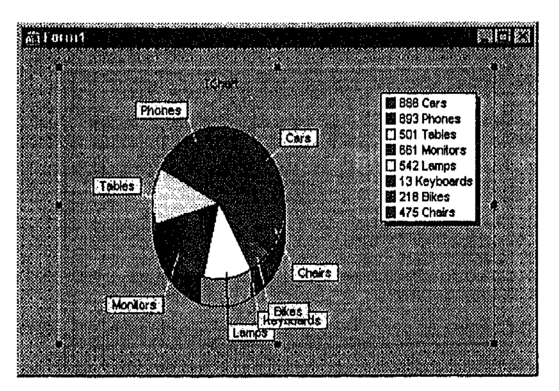
ملحـوظة: تنحـدر الـ TChart class من الـ TPanel class التـابعـة لـ Delphi، ولذلك، يعـتـبـر الـ Chart object هو حـرفـيـاً Panel تم زيادتهـا. هذا يعنى انك تسـتطيع ان تدخل objects اخـرى مـثل الـ Buttons والـ CheckBoxes فى Chart object. انظر البـاب السـابع لمزيد من المعلومات حول الـ Panel.

٤- اضغط يميناً اثناء الاشارة بمؤشر الفأرة داخل الـ Chart. هذا يؤدى الى اظهار القـائمـة المحلية الخـاصـة بالـ Chart. اخـتـر امر الـ ...Edit Chart لفـتح مـحـرر الـ TeeChart. (وهناك طريقة اخرى لفتح المحرر وهى بالضغط مرتين على الـ Chart).

٥- اضغط زر الـ Add للمـحرر لإضافة مجموعة من سلسلة الـ chart. هذا يؤدى الى اظهـار الـTeeChart Gallery dialog الموضح فى شكل (١٨-١). اخـتـر سلسلة. الـ Pie من الـ Gallery، ثم اضـغـط OK واضـغـط Enter لإغـلاق نافذة الـ Gallery dialog. قم بتحريك نافذة المحرر جانباً حتى يمكنك ان ترى الـ form والـ Chart. كمـا يوضح شـكل (١٨-٣)، يعرض الـ TeeChart الـ Chart للنوع المختار مع نقاط بيانات عشوائية مضافة الى الـ Series.

ملحوظة: عندما تقوم بإنشاء chart جديد، تقدم مكتبة الـ TeeChart نقاط بيانات عشوائية حتى ترى نموذج من الناتج النهائى. وهذه البيانات لا يتم عرضها فى وقت التشغيل. عندما تقوم بتشغيل البرنامج، يستخدم الـ chart البيانات الخاصة بك، والتى يمكن ان تأتى من مصادر متنوعة. وهذه الخاصية

تساعدك على تصميم chart حسنة المظهر ، وهى ايضاً طريقة رائعة لكى تتعرف على طريقك فى مكتبة الـ TeeChart .



شكل (١٨-٣): يوضح الـ TeeChart الـ Chart المختارة فى Delphi مع نقاط البيانات العشوائية

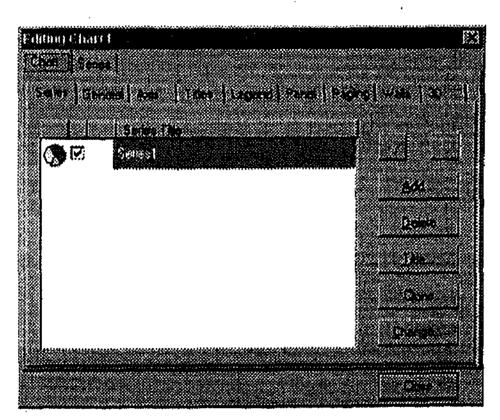
قم بتجربة انواع series اخرى فى الـ TeeChart Gallery [راجع شكل (١٨-١)]. اعد الخطوات السابقة بدءاً من الخطوة رقم ٤ ، واختر زر الـ Add لإضافة series objects إضافية . يمكن للـ chart ان يكون لها series متعددة من نفس النوع- لمقارنة نقاط البيانات مثل اسعار سوق البورصة ، مثلاً، مع خطوط متعددة فى x grid و y . ويستطيع الـ chart ايضاً تجميع series من انواع مختلفة- على سبيل المثال ، لإظهار الـ bar and line charts فى نفس الـ grid ، ربما للمقارنة بين المعلومات التاريخية والحديثة .

استخدم زر الـ Change لمحرر الـ TeeChart لتغيير نوع الـ chart المعروض لـ Series مختارة- على سبيل المثال ، لتغيير bar chart لرسم نقطة . اضغط زر الـ Delete للمحرر لتحذف الـ Series المختارة (هذا يعرض dialog طلب تأكيد فى حالة ان تكون قد ضغطت الـ Delete بطريق الخطأ) . قم بتجربة هذه الخيارات الآن حتى تعتاد عليها .

Note

ملحوظة: كثير من خيارات محرر الـTeeChart تختلف اعتماداً على نوع الـ series التى تختارها . للـ chart اذا كانت الـ objects الموجودة فى هذا الباب تختلف عما تراه على الشاشة فغالباً تكون قد اخترت نوع مختلف من Barchart series بدلاً من رسم الـ Line ، مثلاً .

قم ايضـاً باخـتـيـار بعض page tabs فى مـحـرر الـTeeChart [شكل (١٨-٤)]. هذا يفتح صفحات متنوعة فى نافذة المحرر لتخصيص الـchart. كما ترى، باختيار بعض الجداول، توجد عشرات وعشرات من الخيارات لاختيار الالوان، انماط، الـchart والسمات. جرب هذا: اضغط الـGeneral tab وقم بابطال check box اختيار الـ3D لتحول الـchart الابعاد الى الـchart ثنائى الابعاد. وكذلك جرب هذا: اضغط الـLegend tab وقم بابطال الـVisible check box لإغلاق، والتى تذكر قيم الـpie chart [هذه هى النافذة الخلفية البيضاء الصغيرة الموضحة فى اعلى اليمين فى شكل (١٨-٣)].



شكل (١٨-٤): يوفر محرر الـTeeChart خيارات عديدة لإنشاء وتحرير الـ charts

لاحظ ان محرر الـTeeChart يملك صفين من page tabs. وهذه الـtabs العليا تختار واحداً من المساحتين الكبير تين فى المحرر. وهذه هى:

- **Chart**: اختر الـ tabs العلوى الذى يحمل labeled Chart الخاص بالمحرر، ثم اختر page tabs على الصف الثانى لصنع تغييرات الى خصائص الـ chart. على سبيل المثال، اضغط الـTitles tab وإدخل عناوين chart. لاحظ ان تغييرات قد إنعكست فى الـchart المعروض.

- **Series**: اضغط الـSeries tab فى أعلى المحرر، ثم اختر series من قائمة اللائحة.

يمكنك عندئذ ان تختار tabs الصف الثانى لصنع التغييرات لخصائص الـ series. اضغط الـSeries الآن على الصف العلوى، اختر الـSeries1، اضغط

الـMarks tab على الصف الثانى، واختر واحداً من الازار التالية تحت Style : اختر Value لإظهار قيم فقط البيانات فقط، Percent لإظهار النسب المئوية، Label & Percent لإظهار كلاً من labels النص والنسب المئوية لكل نقطة بيانات، وما الى ذلك.

وهناك طريقة اخرى لاختيار خصائص الـ chart وهى باختيار الـ Chart (اغلق اولاً محرر TeeChart)، ثم استخدم نافذة الـ Object Inspector لتخصيص خصائص الـ chart كما تفعل بـ Delphi الأخرى. ان أغلب الخصائص متاحة فى كلا من محررى TeeChart و الـ Object Inspector . يمكنك استخدام أى method تريده، ولكنك سوف تكتشف ان المحرر اسهل فى الاستخدام بسبب الطريقة التى ينظم بها الخصائص والخيارات فى صفحات. ان الـ Object Inspector يقدم اغلب البنود فى نافذة واحدة كبيرة، وهو الأمر الذى يجعل من الصعب وضع العناصر التى تحتاجها.

Note

ملحوظة: ان محرر الـ TeeChart له خيارات ومميزات عديدة بحيث يصعب تغطيتها فى هذا الباب وحده. ولكن بمتابعة الخطوات المقترحة الموضحة هنا يمكن تعلم كيفية استخدام خيارات اخرى بنفسك. وسوف اشرح ايضاً المزيد عن الخيارات الخاصة للـcharts فى الفصول التالية.

طباعة واصدار الـcharts:

تقدم الـ TeeChart مميزات طباعة ومراجعة object الطباعة النهائى كاملة، والتى يمكنك توفيرها للمستخدم النهائى فى وقت التشغيل. يمكنك استخدام نفس هذه المميزات فى Delphi لطباعة الـ charts. بالإضافة الى ذلك، يمكنك حفظ أى charts فى ملفات جرافيك الدمج فى وثائق اخرى، أو تنفذ عبر الانترنت.

طباعة الـ charts فى Delphi:

لطباعة الـ charts فى Delphi، اضغط يميناً داخل الـ Chart، واختر امر الـ Print Preview... من القائمة المحلية. (اتبع الخطوات الموجودة فى الفصل السابق لإنشاء chart اذا كنت تريد واحدة). يعرض الأمر نافذة الـ TeeChart Print Preview الموضحة فى شكل (١٨-٥).

شكل (١٨-٥): نافذة الـ TeeChart Print Preview

توضح نافذة الـ Print Preview الخريطة كما تظهر عندما يتم طباعتها. استخدم الخيارات المتنوعة لاختيار اتجاه الـ Portrait أو الـ Landscape، ولتعديل الهـوامش، ولتـعـديل تفـاصـيل الـ chart (التـأثيـر بدقـة يعـتـمـد على نوع الـ chart-امسك واترك زر الـ Chart Detail scroll لترى كيف يغير الصورة).

يمكنك ايضاً استـخـدام الفـأرة لضـغط وسـحب خطوط الـهـوامش المنقـوطة وتحرك الـ chart عبر الصفـحة. جرب هذا الآن-اضغط الفأرة داخل نافـذة مراجعة الـ object النهائي واسحبها لتغير الـ chart وحجمه.

عندما تنتهي من ضبط الـ chart واختـيـار الخـيـارات (ايضـاً اضـغط زر الـ ...Printer Setup لاختيار خيارات الاخراج الخاصة بطابعتك)، اضـغط زر الـ Print للطبـاعـة. يمكنك عندئذ ضـغـط زر الـ Close لإغـلاق نافـذة الـ Print Preview.



تحذير: عندما تغلق نافذة الـ Print Preview، ترجع كل المواصفات الى البدائل الافتراضية حسب النظام. لا تغلق النافذة الا بعد الانتهاء من طباعة الـ Chart.

طباعة الـ Chart وقت التشغيل:

يستطيع مسـتـخـدمـوا برنامجك ان يظهـروا نافـذة الـ Print Preview، ويستـخدموا كل الخيارات الموضحة في الفصل السابق. لتجربة هذا، اضف الـ

Button objectform، واضغطه مرتين لإنشاء الـ OnClick event. الخاص به .
إنتقل الى نافذة محرر الـDelphi code، إبحث عن كلمة الـ implementation الاساسية . واضف عبارة uses كما يلى :

```
implementation
{$R *.DFM}
uses
  TeePrevi;
```

هذا يجعل (TeeChart Print Preview) متاحة للـ module . يمكنك الآن برمجة الـ Button event handler كما يلى :

```
procedure TForm.Button1Click(Sender: TObject);
begin
    ChartPreview(Form1, Chart1);
end;
```

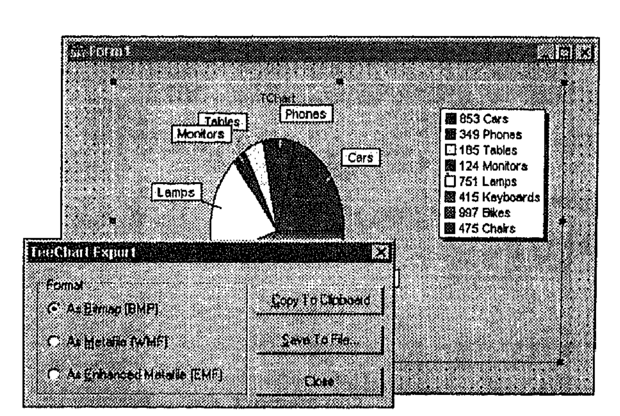
قم باستدعاء الـ ChartPreview procedure الخاص الـTeePrevi unit وقم بتمرير الـ chart form (Form1 هنا) و (Chart1) على انها arguments هذا يعرض نافذة الـ Print Preview، الموضحة فى شكل (١٨-٥). وتتولى الـunit تماماً كل خيارات الطباعة-لاحاجة لكتابة أىcloses اخرى. عندما يغلق المستخدم النافذة، يستمر البرنامج بشكل طبيعى .

تنفيذ الـcharts فى Delphi:

تنفيذ الـ charts فى Delphi، اضغط يميناً داخل الـ Chart objects لفتح القائمة TeeChart (اذا لزم الأمر، قم بإنشاء chart). اختار امر الـ Export ...Chart الذى يعرض نافذة الـ dialog الموضحة فى شكل (١٨-٦).

ان الـ TeeChart Export dialog يقدم ثلاثة خيارات لنسخ chart الى الـ Windows clipboard، أو لحفظ الـ chart فى ملف قرص. و الـ formats الثلاث هى :

- **As Bitmap (BMP):** استخدم هذا الخيار لحفظ الـ chart فى ملف bitmap، وحجم هذا الملف يعتمد على حجم الـ chart .



شكل (١٨-٦):استخدم الـ TeeChart Export dialog لنسخ صورة chart الى الـ Windows clipboard، أو لحفظها فى ملف صورة

- **As Metafile (WMF):** استـخـدم هذا الخـيـار لحـفـظ الـ chart فى ملف meatfile. وهذا ليس صـورة، ولكن مجموعة من الاوامر التى ترسم صـورة تشبه بالضبط البرنامج الأصلى. والـ metafile تستخدم فقط مساحة قليلة على القرص، ويسهل قياسها بابعاد عرض جديدة. استخدم هذا الخيار اذا كانت صورك تحتاج ان يتم رؤيتها من جانب مستخدمى الـ Windows 3.1.

- **As Enhanced Metafile (EMF):** استـخـدم هذا الخيار لحفظ chart فى ملف enhanced metafile تم زيادته. هذا يشبه ملف الـ metafile المعيارى، ولكن يمكن رؤيتـه فـقط من قبل مـسـتـخـدمى الـ Windows 95 و 98 و NT. وتعتبر ملفات enhanced metafile التى تم زيادتها افضل من جوانب كثيرة، بما فى ذلك الدقة فى إعادة الانتاج والقياس، عن ملفات الـ metafile المعيارية. ان ملفات enhanced metafile التى تم زيادتها تستخدم إحداثيات ذات ٣٢ بت؛ الـ metafile المعيارية تستخدم احداثيات ذات ١٦ بت. للحصول على افضل النتائج، استخدام ملفات enhanced metafile الزائدة اينما كان ممكناً.

بعـد اختيار واحداً من خيارات التنسيق الثلاثة، اضـغط زر الـ Copy to Clipboard لنسخ الصورة الى الـ Windows clipboard. يمكنك عندئذ ان تنتقل الى برنامج آخر مثل محرر النص وتلصق الصورة فى وثيقة (بافتراض ان البرنامج يستطيع التعامل مع الصور فى الـ form المختارة). اضغط زر الـ ...Save to File

لإظهار الـdialog الذى تستطيع استخدامه لإدخال اسم الملف المستهدف، ولاختيار دليل الاخراج.

تنفيذ الـ charts فى وقت التشغيل:

لتنفيذ الـ charts فى وقت التشغيل، استدع واحداً من الـ TChart methods: الثلاثة التالية :

```
SaveToBitmapFile(const Filename: String);
SaveToMetafile(const Filename: String);
SaveToMetafileEnh(const Filename: String);
```

والى كل method، قم بتمرير اسم الملف، والذى قد يشمل معلومات عن اسم المسار والمحرك. يجب ان يتناسب امتداد اسم الملف مع نوع الملف- فهذا لا يتم إضافته بصورة تلقائية. على سبيل المثال، لحفظ الـ chart للـ Windows bitmap، استخدم عبارة مثل :

```
Chart1.SaveToBitmapFile('C:\Test.bmp');
```

Note

ملحوظة: بعض النسخ من الـ online help للـ TeeChart تسمى الـ procedure السابق بـ SaveToBitmap وهى تسمية خاطئة. والاسم الصحيح هو .SaveToBitmapFile.

استدع الـ functions الاخرى بنفس الطريقة. تقوم العبارتان التاليتان بحفظ الـ charts كملف metafile و كملف enhanced metafile Windows تم زيادته. كما سبق، إنها مسئوليتك ان تحدد امتداد اسم الملف الصحيح لنوع الملف الذى تنشئه :

```
Chart1.SaveToMetafile('C:\Test.wmf');
Chart1.SaveToMetafileEnh('C:\Test.emf');
```

فهم مصادر بيانات الـ chart:

كما اوضحت الفصول السابقة، فمن السهل إنشاء chart مع مكتبة الـ TeeChart. ولكن اختيار افضل نوع لـ series chart معينة من نقاط البيانات هو عادة اصعب جزء فى مهمة عمل الـ chart. لإنشاء chart ناجحة فإنك تحتاج ان

تفكر جيداً فى مصدر بياناتك، والمعلومات التى يقدمها وهدفك من توفير تمثيل بصرى لتلك المعلومة.

لكى نساعدك فى البدء فى تطوير الـ charts حقيقية باستخدام ثلاثة انواع اساسية من مصادر البيانات:

- **Program data**: استخدام مصدر البيانات هذا لإنشاء charts باستخدام بيانات تكتبها مباشرة فى الـ source code للبرنامج. ويمكن تثبيت هذه البيانات لعرض chart لمعلومات تاريخية مثلاً- أو يمكن حسابها باستخدام functions البرنامج.

- **File data**: استخدام مصدر البيانات هذا لإنشاء chart بيانات مخزنة فى ملف قرص. ويرجع نوع الملف إليك. فقد يكون ملف لبيانات تم إنزالها من الانترنت، أو يمكن ان يكون ملف نص ذو معلومات تدخلها انت أو تقصها وتلصقها من وثيقة اخرى.

- **Database data**: استخدام مصدر البيانات هذا لإنشاء charts باستخدام بيانات تم تحميلها من Table قاعدة بيانات. وكحد ادنى، إنك تحتاج ايضاً لاستخدام Table components و DataSource لتوصيلها الى قاعدة بيانات. (انظر الباب السابع عشر لمزيد من المعلومات عن components قاعدة بيانات Delphi).

يوضح الفصل التالى كيفية برمجة الـ charts باستخدام بيانات من كل من هذه المصادر الثلاثة. والبرامج فى كل فصل ايضاً توضح تقنيات charts إضافية.

charts من بيانات برنامج:

ان إحدى الطرق لتوفير بيانات لـ chart هى كتابتها مباشرة فى الـ source code للبرنامج. أو، يمكنك استخدام هذا الـ method لعرض معلومات تم توفيرها بواسطة procedures متنوعة مثل تشغيل functions النظام. لهذه الانواع من البيانات، استخدم الـ Chart على الـ Additional palette.

قم باسقاط Chart على form، اضغطه مرتين واختر ...Edit Chart لفتح محرر الـ TeeChart . اضغط زر الـ Add لإنشاء واحداً أو أكثر من الـ Series



ولاختيار نوع الـ series objects الذى يتم عرضه. يمكنك ايضاً اختيار خيارات متنوعة للـ charts فى هذا الوقت.

بعد الانتهاء من تصميم الـ charts، اضف عبارات برنامج لتزودها ببيانات حقيقية. لإدخال البيانات، استدع الـAdd methods، أو AddY، أو AddX، أو AddXY للـ series objects الخاصة بك لإضافة نقاط بيانات. على سبيل المثال، تضيف العبارات التالية AValue (بافتراض انها متغير مزدوج) للـ Series1، وتحدد labelنقطة البيانات هذه بالـ string 'Value',، وتحدد ان الـ Chart يجب ان يختار لوناً ملائماً:

```
Series1.Add(AValue, 'Value', clTeeColor);
```

وبدلاً من الـ clTeeColor، يمكنك استخدام أى قيمة TColor مثل الـ clBlue أو الـ clRed، أو يمكنك تحديد الوان بالقيم العشرية. (لمزيد من المعلومات راجع استخدام الـ TColor فى online help الخاصة بـ Delphi، وكذلك انظر الباب الثالث عشر، "تطوير تطبيقات الجرافيك").

On the CD-ROM **على القرص المدمج:** ان التطبيق MemInfo على القرص المدمج فى دليل Source\MemInfo يوضح كيفية إضافة بيانات برنامج الى chart. يوضح شكل (١٨-٧) البرنامج، الذى يعرض الـ bar chart لذاكرة البرنامج تستخدم كما هى واردة من قبل الـSystem.GetHeapStatus function الخاصة بـ Delphi. لتشغيل البرنامج، قم بتحميل ملف مشروع الـ MemInfo.dpr فىDelphi واضغط F9. عندما تظهر نافذة البرنامج، اضغط زر الـ ShowInfo لعرض قيم استخدام ذاكرة متنوعة. يمكنك إضافة form هذا البرنامج الى مشروعاتك لإزالة مشكلات الذاكرة، أو لمجرد مراجعة استخدام الذاكرة. توضح القائمة (١٨-١) الـ source code للبرنامج.

القائمة (١٨-١): MemInfo\Main.pas

```
unit Main;

interface

uses
```



```
    Windows, Messages, SysUtils, Classes, Graphics,
    Controls,
      Forms, Dialogs, StdCtrls, TeEngine, Series, ExtCtrls,
      TeeProcs, Chart, Buttons;

type
  TMainForm = class(TForm)
        Chart1: TChart;
            Button1: TButton;
            Series1: TBarSeries;
            BitBtn1: TBitBtn;
          . Label1: TLabel;
            procedure Button1Click(Sender: TObject);
          private
    { Private declarations }
      public
    { Public declarations }
      end;

var
  MainForm: TMainForm;

implementation

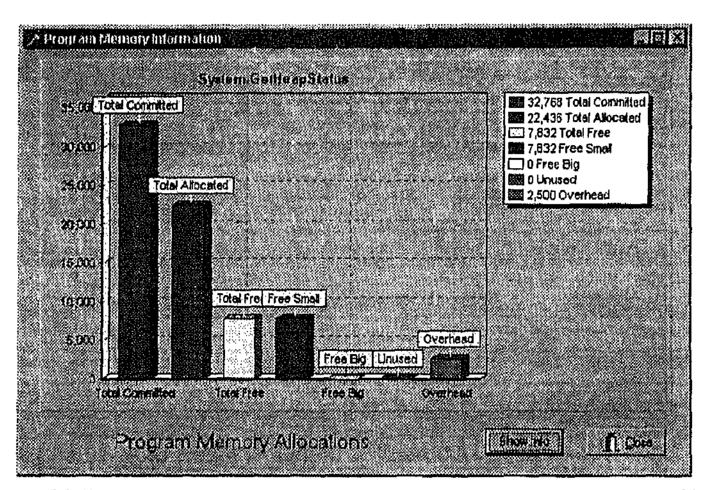
{$R *.DFM}

var
  HeapStatus: THeapStatus;

procedure TMainForm.Button1Click(Sender: TObject);
begin
  HeapStatus := System.GetHeapStatus;
      with Series1, HeapStatus do
      begin
//    Add(TotalAddrSpace, 'Total address space',
clTeeColor);
```

```
//    Add(TotalUncommitted, 'Total Uncommitted',
clTeeColor);
    Add(TotalCommitted, 'Total Committed', clTeeColor);
        Add(TotalAllocated, 'Total Allocated', clTeeColor);
        Add(TotalFree, 'Total Free', clTeeColor);
        Add(FreeSmall, 'Free Small', clTeeColor);
        Add(FreeBig, 'Free Big', clTeeColor);
        Add(Unused, 'Unused', clTeeColor);
        Add(Overhead, 'Overhead', clTeeColor);
      end;
end;

end.
```



شكل (١٨-٧): تطبيق MemInfo يوضح استخدام البرنامج للذاكرة وكذلك يعرض كيفية إضافة بيانات برنامج الى الـ Chart

للحصول على قيم استخدام الذاكرة، يستدعى برنامج الـ MemInfo والـ function الـ GetHeapStatus فى الـ System unit الخاصة بـ Delphi. تقوم الـ بإرجاع سجل من نوع الـ THeapStatus، وهو معرف كمتغير فى قطاع الـ implementation للـ MemInfo. تملأ العبارة التالية سجل الـ HeapStatus بقيم مثل الـ TotalCommitted والـ TotalAllocated، والتى تمثل استخدام ذاكرة البرنامج. (راجع الـ online help الخاصة بـ Delphi لمعرفة المزيد من المعلومات من كل متغير فى سجل الـTHeapStatus).

```
HeapStatus := System.GetHeapStatus;
```

لإضافة القيمة التى تم إدخالها الى الـ Series1 الخاص بالـ chart، يقوم الـ OnClick الخاص بزر الـ Show Info باستدعاء الـ Add method . على سبيل المثال، هذه العبارة تضيف قيمة الـ TotalFree الى الخريطة :

```
Add(TotalFree, 'Total Free', clTeeColor);
```

هذا يؤدى الى إنشاء bar جديد فى القيمة الموجودة فى الـ HeapStatus الـ TotalFree يحدد labels للـ bar من خلال string . وقيمة اللون clTeeColor تخبر الـ color ان يختار اللون بصورة تلقائية لنقطة البيانات هذه. ويتم إضافة نقاط البيانات الأخرى بنفس الطريقة.

ملحوظة: لا يعرض الـ MemInfo لإثنين من قيم الـ GetHeapStatus، وهى TotalAddrSpace و TotalUncommitted . فهذه تعتبر قيم نظام عامة، والتى لا ترتبط مباشرة بقيم ذاكرة البرنامج الأخرى الموجودة فى الخريطة. وهذا مثال جيد على متى تكون المعلومات الكثيرة جداً أكثر ضرراً، لذا فقد حذفت العبارتين الواقعتين بالقرب من بداية للـ Button.

ان الـ Add method المستخدم فى الـ MemInfo يعتبر مناسبا للـ bar charts البسيطة. وهو ايضاً ملائم لإنشاء الـ pie charts لان هذه توضح البيانات كنسب مئوية من الكل-لمقارنة الكثافات السكانية للمدن، مثلاً، أو لإظهار عدد المنتجات فى مخزن مصنع. والانواع الاخرى من الـ charts مثل راسمات الخطوط تنتقى المعلومات المرسومة على grid احداثيات الـ x والـ y. لهذه الانواع من الـ charts احداثيات قد يكون لديك فى بعض الاحيان قيم x أو y، أو قد يكون لديك كليهما. لإضافة نقاط البيانات لهذه الانواع من الـ series، استدع الـ AddXY procedures أو AddY أو الـ AddX. تضيف العبارة التالية YValue (قيمة مزدوجة من المفترض ان تكون معرفة من قبل البرنامج)، وتعطى محور الـ Y بطاقة تحمل string 'Y-Label'، وتلون العرض الخاص بهذه البيانات باللون الاحمر :

```
Series1.AddY(YValue, 'Y-Label', clRed);
```

احداثية الـ x لنقطة البيانات هذه يتم حسابها بصورة تلقائية باستخدام القيم المختارة مع page خيارات الـ ChartlAxis الخاصة بمحرر الـ TeeChart . بنفس الطريقة ، اذا كان لديك قيمة x، استدع الـ AddX لإنشاء نقطة بيانات جديدة للـ series :

```
Series1.AddX(XValue, 'X-Label', clWhite);
```

اخيراً، اذا كان لديك إثنين من القيم للـ chart كمواضع x و y في الـ grid ، استدع الـ AddXY باستخدام عبارة مثل هذه :

```
Series1.AddXY(XValue, YValue, 'X-Label', clBlue);
```

Charts من بيانات ملف:

ان مصدر البيانات الذى يعد أكثر إنتشاراً للـ charts يأتي من معلومات مخزونة في ملفات والمشكلة الرئيسية في قراءة هذه البيانات هي معرفة format الملف . فقد يكون ملف نص تم انزاله من على شبكة الانترنت ، أو قد يحتوى على قيم مخزنة في صورة binary اذا عرفت أو استطعت تحديد format الملف ، فيمكنك كتابة procedure لتحميل البيانات في الذاكرة ثم استدعاء الـ Add ، أو الـ AddY ، أو الـ AddX ، أو الـ AddXY methods لإدخال نقاط بيانات في chart series.

على القرص المدمج: وكعرض واقعي للمشكلات المتضمنة في قراءة ملفات بيانات ، فإن التطبيق MayTemp على القرص المدمج في دليل Source\MayTemp يعرض رسماً بيانياً للبيانات الرقمية التي قمت بانزالها من على موقع شبكة الانترنت Internet Web site http:// www-mfl.nhc.noaa.gov . هذا يتوصل الى مكتب الـ National Weather Service (أو الخدمة الوطنية للمناخ) في ميامي بولاية فلوريد، والتي توفر تنبؤات عن حالة الطقس، charts ومعلومات اخرى. في هذه الحالة، كنت مهتماً. ببيانات المناخ في منطقة Key West بولاية فلوريدا لشهر مايو، وبخاصة متوسط درجات الحرارة العظمى والصغرى. وهذه البيانات موجودة في ملف /climate/Key-West_May_Climat.html والموضح في الـ form الاساسية

فى شكل (١٨-٨) كـمـا هو مـعـروض على نظامى باسـتـخـدام الـ Internet Explorer. كما ستدرك ، ان الـ Internet محملة ببيانات ممتعة مثل هذه .

Key West May Climat - Microsoft Internet Explorer - [Working Offline]

http://www.mfl.nhc.noaa.gov/climate/Key-West_May_Climat.html

KEY WEST, FLORIDA

MAY Daily Normal and Record Temperatures

Monthly Normal and Record Precipitation

January 1871-1996

Date		Normal Low @		Year	Record Low	Year	Coldest Maximum	Year		Year
1	83	74	90	1873	64	1992	74	1988	80	1964
2	84	74	89	1953	63	1977	76	1925	79	1964
3	84	75	91	1873	66	1992	76	1897	80	1994
4	84	75	90	1956	66	1925	74	1989	80	1984
5	84	75	91	1873	64	1992	72	1889	80	1991
6	84	75	90	1946	67	1992	75	1911	81	1984

شكل (١٨-٨): بيانات المناخ الاصلية من الـNational Weather Service لمنطقة Key West بولاية Florida لشهر مايو (وبطاقة January (يناير) هى خطأ بالتأكيد) من ١٨٧١ الى ١٩٩٦

Note

ملحوظة: ان القراء ذوى الابصار الحادة قد يلحظون ان البيانات الاصلية تحمل البطاقة "MAY" ولكن لها عنوان جانبى - "January 1871 1996." . من الواضح ان هذا خطأ (فـهـذه بيـانات May، وليس January)، وهذا ما ارسلت به لـ NWS. وهذا ايضاً يوضح ان، عند إنزال بيانات من على الانترنت، من الافضل ان تختبر صلاحية المعلومات التى يتم تلقيها.

On the CD-ROM

على القرص المدمج: ان الحصول على البيانات الخام كان هو الجزء السهل- ان نقل هذه البيانات من على الانترنت الى ملف ثم الى برنامج Delphi للعرض كرسم بيانى بصرى يتطلب مزيداً من العمل. لقد اخترت اولاً البيانات الخام فى نافذة الـ Internet Explorer ، ضغطت Ctrl+C



لنسخ النص الى لوحة الملصقات، ثم قمت بفتح الـWindows Notepad utility. ان ضغط Ctrl+V يلصق المعلومات الى الـ Notepad . بعد حذف النص الزائد قمت بحفظ البيانات فى ملف نص يحمل اسم KeyWestMayClimate.txt . وهذا الملف، والموضح هنا فى القائمة (١٨-٢)، موجود على القرص المدمج فى دليل Source\Data . (وكذلك موجود فى هذا الدليل كمرجع ملف الـ Key-West_May_Climat.html، والذى يمكنك تحميله فى متصفح الانترنت الخاص بك اذا اردت التدرب على ارسال البيانات بنفسك).

القائمة (١٨-٢): Data\KeyWestMayClimate.txt

```
1 83 74 90 1873 64 1992 74 1988 80 1964
2 84 74 89 1953 63 1977 76 1925 79 1964
3 84 75 91 1873 66 1992 76 1897 80 1994
4 84 75 90 1956 66 1925 74 1989 80 1984
5 84 75 91 1873 64 1992 72 1889 80 1991
6 84 75 90 1946 67 1992 75 1911 81 1984
7 84 75 89 1956 66 1954 76 1903 80 1991
8 84 75 91 1873 65 1988 74 1898 80 1991
9 84 75 92 1873 64 1992 76 1928 80 1978
10 85 75 90 1873 66 1960 75 1891 80 1995
11 85 76 90 1873 64 1944 77 1891 80 1995
12 85 76 90 1926 68 1944 77 1900 80 1996
13 85 76 90 1995 66 1888 76 1900 81 1995
14 85 76 90 1878 65 1917 74 1900 81 1995
15 85 76 90 1991 68 1932 76 1917 81 1994
16 85 76 90 1991 66 1951 76 1977 82 1994
17 85 76 92 1878 66 1951 78 1880 81 1989
18 85 76 92 1881 67 1951 77 1904 81 1995
19 85 77 92 1881 68 1917 77 1904 82 1995
20 86 77 91 1886 68 1911 76 1875 81 1995
21 86 77 91 1935 68 1901 74 1875 80 1885
22 86 77 92 1935 68 1940 76 1921 82 1985
23 86 77 92 1873 69 1940 77 1892 80 1995
```

```
24 86 77 92 1873 68 1900 78 1883 82 1985
25 86 77 91 1989 70 1913 78 1982 82 1995
26 86 77 91 1989 68 1992 81 1879 81 1995
27 86 77 90 1989 70 1894 79 1880 81 1995
28 86 77 92 1953 68 1916 77 1901 81 1924
29 86 77 91 1952 68 1926 78 1918 81 1995
30 86 78 92 1949 67 1932 76 1960 82 1915
31 87 78 93 1881 70 1934 79 1960 82 1985
```

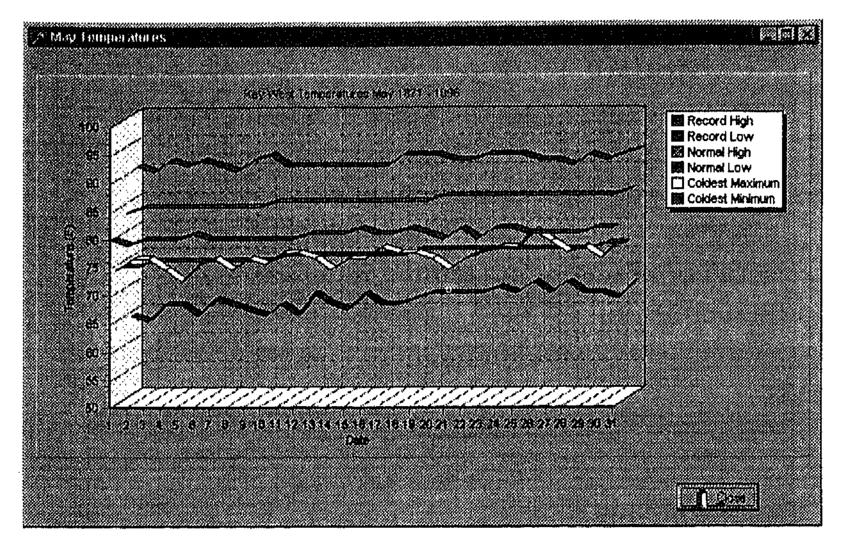
على القرص المدمج: ان البيانات الموضحة هنا عبارة عن text form تعتبر نموذجية. فى هذه الحالة، تخبرنا البيانات الاصلية عن صفه القيم، لذا لا يصعب قراءتها فى متغيرات برنامج. ولان هذا يعتبر ملف نص، فإن الـ Read procedure الخاص بالـ Object Pascal يعد افضل طريقة لتحميل البيانات وعرضها بصرياً. يوضح شكل (١٨-٩) الـ chart التام، والمعروضة من خلال تطبيق الـ MayTemp على القرص المدمج فى دليل Source\MayTemp. قم بتحميل ملف مشروع الـ MayTemp.dpr و Delphi واضغط F9 للـ compile والتشغيل. توضح القائمة (١٨-٣) الـ source code للبرنامج.

ملحوظة: يقرأ برنامج الـ MayTemp ملف البيانات KeyWestMayClimate.txt الموجود فى دليل الـ Source\Data على القرص المدمج. اذا فشل البرنامج فى التشغيل بشكل سليم، فتأكد من ان الدليل الحالى هو Source\MayTemp، أو قم بتعديل اسم المسار فى ثابت الـ FileName الخاص بالبرنامج.

يستخدم الـ MayTemp تقنيات Object Pascal معيارية لفتح ملف نص وقراءة معلوته. لفعل هذا فى وقت التشغيل، يستخدم البرنامج الـ OnActivate للـform. هذا يضمن ان تكون البيانات قد تم تحميلها وإضافتها بطريقة سليمة الى chart series objects، عندما يبدأ البرنامج، ولكن قبل ان تصبح النافذة مرئية.

ان TMainForm.FormActivate، يعرف متغيرات صحيحة لكل قيمة بيانات-متغير واحد لكل بند على كل صف [انظر شكل (١٨-٢)]. لقراءة بيانات الملف فى هذه المتغيرات، يعرف الـ procedure ايضاً متغير ملف:

```
var
  F: TextFile;
```



شكل (١٨-٩): الـ chart التام، توضح بيانات درجات الحرارة من المعلومات الخام الموجودة بالقائمة (١٨-٢)

القائمة (١٨-٣): MayTemp\Main.pas

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics,
   Controls,
    Forms, Dialogs, TeEngine, Series, ExtCtrls, TeeProcs,
   Chart,
    StdCtrls, Buttons;

type
  TMainForm = class(TForm)
      Chart1: TChart;
         Series1: TLineSeries;
         Series2: TLineSeries;
```

```
        Series4: TLineSeries;
          Series5: TLineSeries;
          Series6: TLineSeries;
          Series3: TLineSeries;
          BitBtn1: TBitBtn;
          procedure FormActivate(Sender: TObject);
         private
   { Private declarations }
    public
   { Public declarations }
         end;

var
 MainForm: TMainForm;

implementation

{$R *.DFM}

{ Change the following path name if your data file is
 in another location. The path name is relative to the
    current directory, which is assumed to be at the
    same level as \Data\. }

const
 FileName = '..\Data\KeyWestMayClimate.txt';

{ The following procedure is called when the form is first
 activated. At that time, the program opens the data file,
    reads its values, and adds them to the chart's series
    objects. }

procedure TMainForm.FormActivate(Sender: TObject);
var
 F: TextFile;              // File variable
    Date: Integer;            // First column in file data
```



```
    NormalHigh : Integer;        // Next column
     NormalLow : Integer;         // and so on ...
     RecordHigh : Integer;
     RecordHighYear : Integer;
     RecordLow : Integer;
     RecordLowYear : Integer;
     ColdestMaximum : Integer;
     ColdestMaximumYear : Integer;
     WarmestMaximum : Integer;     // ... down to the
     WarmestMaximumYear : Integer;  // Last column in
    file data
begin
  AssignFile(F, FileName);  // Initialize file variable
     Reset(F);              // Open the file
     while not Eof(F) do      // Loop until the end of the file
     begin
    Read(F,               // Read one row of data
          Date,              // into the individual variables.
            NormalHigh,
            NormalLow,
            RecordHigh,
        (continued)
        Listing 18-3 (continued)
            RecordHighYear,
            RecordLow,
            RecordLowYear,
            ColdestMaximum,
            ColdestMaximumYear,
            WarmestMaximum,
            WarmestMaximumYear);  // End of Read
        statement

    { One row of file data has been loaded at this point. The
           following statements add the data points to each
        of the
            line chart's six series objects. The empty string
```



```
arguments can be used to label data points. These
   strings aren't used here because the X axis of
this
   sample chart already shows day values (1, 2, ...,
31). }

Series1.AddXY(Date, NormalHigh, '', clTeeColor);
    Series2.AddXY(Date, NormalLow, '', clTeeColor);
    Series3.AddXY(Date, RecordHigh, '', clTeeColor);
    Series4.AddXY(Date, RecordLow, '', clTeeColor);
    Series5.AddXY(Date, ColdestMaximum, '',
  clTeeColor);
    Series6.AddXY(Date, WarmestMaximum, '',
  clTeeColor);

 end;
end;

end.
```

هناك عبارتان تبدأن هذا المتغير باستخدام الـ FileName string الثابت، ويفتحان الملف للإعداد لقراءة بياناته:

```
AssignFile(F, FileName);
Reset(F);
```

بعد هذه الخطوات، تستمر عبارة while فى التنفيذ حتى يتم تحميل كل صفوف البيانات فى متغيرات البرنامج. وهذه الجزئية، على سبيل المثال، توضح كيف يقوم البرنامج بتحميل القيمة الاولى فى كل صف، مثلاً البيانات (١، ٢، . . . ٣١):

```
while not Eof(F) do
begin
 Read(F,
    Date,
...
end;
```

يتم تحميل المتغيرات الاخرى بطريقة مشابهة، مع عبارة Read واحد، وذلك بذكرها منفصلة بواسطة الفصلات.

Tip **فكرة:** استخدم الـ Read procedure الخاص بالـ Pascal لقراءة قيم فردية على نفس خط النص الاصلى فى متغيرات منفصلة. استخدم الـ Readln لقراءة سطر باكمله كـstring.

بعد تحميل كل صف بيانات، يقوم البرنامج بادخال أغلب القيم فى chart series. فى هذه الحالة، يكون لكل خط معروض فى الرسم البيانى النهائى، [انظر شكل (١٨-٩)]. يتم اضافة كل نقطة بيانات باستدعاء الـ TeeChart AddXY method على سبيل المثال، العبارتان التاليتان تدخلان قيم NormalHigh و NormalLow فى series objects الخاصة بهما على التوال:

```
Series1.AddXY(Date, NormalHigh, '', clTeeColor);
Series2.AddXY(Date, NormalLow, '', clTeeColor);
```

ان الأولى Date argument ، وهى نفسها لكلاً من نقطتى البيانات- يمثل محور الـ x فى هذه الحالة يوم فى الشهر الـargument التالية هى القيمة التى يجب استخدامها. لنقطة البيانات هذه. ان الـ null string لاحاجة له لان chart يعرض بالفعل قيم درجات حرارة على طول محور الـ y. و argument الاخيرة تخبر الـ chart ان يستخدم اللون الذى تم برمجته فيما سبق لنقطة البيانات هذه.

Tip **فكرة:** ان البيانات العشوائية الخاصة بالـ TeeChart للـ chart الموجود فى تطبيق الـMayTemp لا تنتج الـ chart فى وقت التصميم. وهذا يحدث لان القيم العشوائية تقع خارج نطاق إحداثيات الـ x والـ y المحددة والتى اريد استخدامها للـ chart. لإغلاق توليد البيانات العشوائية، قمت بفتح محرر الـ TeeChart، وضغطت الـ Series page tab، واخترت كل series object من قائمة اللائحة. لكل series، قمت باختيار الـ Data Source page tab ثم الـ No Data من قائمة الائحة لخيارات مصدر البيانات. قد تريد ان تفحص هذه المنطقة من محرر الـ TeeChart فى حالة ما اذا احتجت ان تغلق توليد البيانات العشوائية لـcharts.

Charts من بيانات قاعدة البيانات:

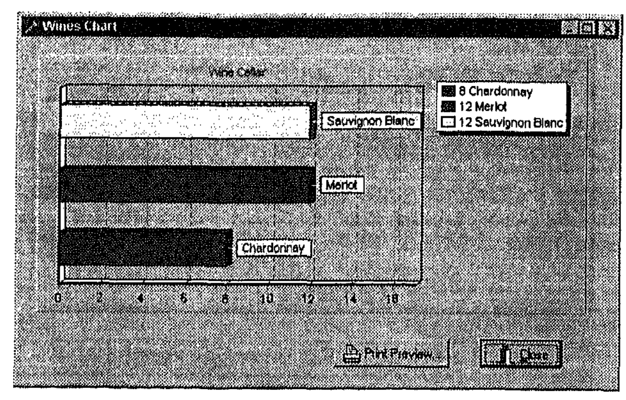
والطريقة الثالثة والاخيرة للحصول على بيانات chart هى تحميل قيم حقل من جداول قاعدة بيانات. وهذه التقنية تستخدم components مثل الـ Table والـ DataSource، الموضحة فى الباب السابع عشر، لفتح Table قاعدة بيانات. ولكن بدلاً من الـ Chart component المستخدم حتى الآن، فلتحميل بيانات من جدول قاعدة بيانات، استخدم الـ DBChart component الموجود على الـ Data Controls palette الخاصة بـ Delphi. يعمل الـ DBChart تماماً مثل الـ Chart، باستثناء ان بياناته تأتى من جدول قاعدة بيانات. وكل السمات والخيارات الاخرى هى نفسها المذكورة فى الفصول السابقة.

On the CD-ROM

على القــرص المدمج: على القــرص المدمج، يوضح تطبـيق الـ WinesChart كيفية إنشاء chart من معلومات قاعدة بيانات. يوجد هذا البــرنامج فى دليل الـ Source\WinesChart. يوضح شكل (١٨-١٠) عرض البـرنامج باسـتخـدام قـاعـدة بيـانات الـ Wines فى دليل الـ Source\Data على القرص المدمج. توضح القائمة (١٨-٤) الـ source code للبرنامج.

Note

ملحـوظة: اذا لم يعـمل الـ WinesChart على نظامك، فـإن السـبب غـالبـاً يكون نقص الـ WINES Alias name. لإنشـاء هذا الـ Alias name، اختـر Tools|Database Desktop، ثم فى برنامج الحـاسب المكتـبى، اخـتـر Tools|Alias Manager.... اضـغط زر الـ New، وإدخل WINES عـلى انـه الـ Alias name. حــــدد الـدلـيـل المرتـبـط بـدلـيـل Source\Data\Wines من عـلى القــرص المدمج. اضـغـط Keep New ثم اختر OK. اجب بـ Yes، عندما تسأل عما اذا كنت تريد حفظ الـ Alias name عامة. يجب ان يكون قادراً الآن على تشغيل الـ WinesChart بنجاح. انظر الباب السابع عشر لمزيد من المعلومات حول تحديد واستدام الـ Alias name لقاعدة البيانات.



شكل (١٨-١٠): عرض وقت التشغيل لبرنامج الـ الـ WinesChart يوضح كيفية قراءة معلومات من جدول قاعدة بيانات وعرضها كـ chart

القائمة (١٨-٤): WinesChart\Main.pas

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics,
Controls, Forms, Dialogs,
  StdCtrls, Buttons, TeEngine, Series, ExtCtrls, TeeProcs,
Chart, DBChart,
  Db, DBTables;

type
  TMainForm = class(TForm)
      Table1: TTable;
        DataSource1: TDataSource;
        DBChart1: TDBChart;
        Series1: THorizBarSeries;
        BitBtn1: TBitBtn;
        BitBtn2: TBitBtn;
        procedure BitBtn2Click(Sender: TObject);
  private
```

```
        { Private declarations }
         public
        { Public declarations }
         end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}

uses
  TeePrevi;

procedure TMainForm.BitBtn2Click(Sender: TObject);
begin
  ChartPreview(MainForm, DBChart1);
end;

end.
```

لإنشاء تطبيق الـ WinesChart إتبع الخطوات التالية :

١- قم باضــــافـــة Table و DataSource على form، مع DBChart object . كـذلك إدخل إثنين من الازرار، تحمل ...labeled Print Preview فى Close كـمــا فى شكل (١٨-١١)، الذى يوضح نافــذة الـ form المنتـهـيــة فى Delphi .

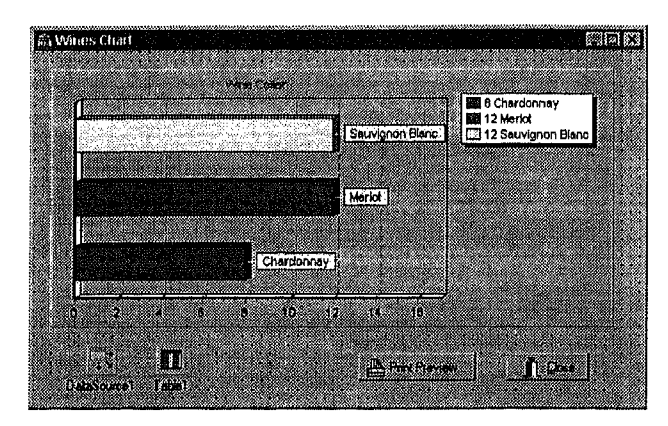
٢- حـدد خـاصـيـة الـDatabaseName للـ Table1 بـ WINES حـدد الـ TableName بـ Wines.db . ثم، غير الـ Active لتصبح True .

٣- حدد خاصية الـ DataSet للـ DataSource1 بـ Table1 .

٤- اختـر DBChart1، واضـغط الفـأرة يميناً لإظهـار القـائمـة المحلية للـ TeeChart . اختر ...Edit Chart لعرض محرر الـ TeeChart .



٥- اضغط زر الـ Add للمحرر واختر Horiz. Bar chart series . اضغط OK . يجب ان ترى بيانات عشوائية معروضة فى chart .



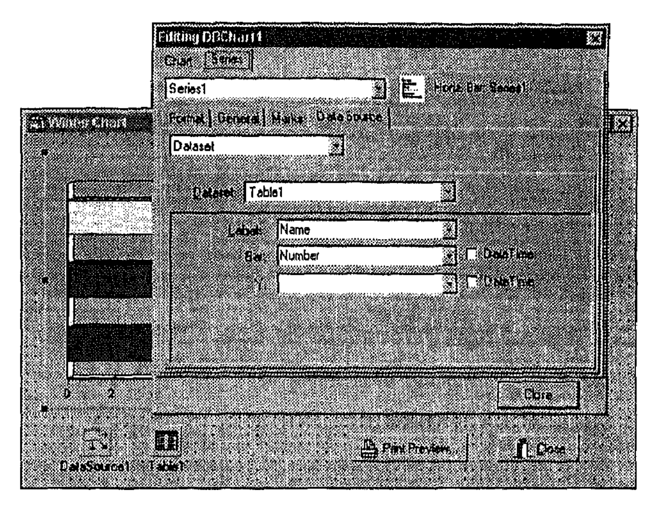
شكل (١٨-١١): عرض وقت التصميم لنافذة الـ form الـ WinesChart ايضاً يوضح الـ DataSource1 والـ Table1 التى تربط البرنامج بقاعدة بيانات الـ Wines

٦- لربط ايضاً chart بقاعدة بيانات WINES ، اختر الـ Series page tab لمحرر الـ TeeChart فى أعلى النافذة . ولان هناك series objects واحدة ، فتكون قد تم اختيارها بالفعل (فى برامجك ، اذا استخدمت series objects متعددة ، اختر كل واحد من قائمة اللائحة ثم اعد الخطوة التالية) .

٧- اضغط الـ Data Source tab فى محرر الـ TeeChart . افتح قائمة اللائحة تحت الصف الثانى من page tabs واختر مدخل الـ Dataset . بنفس الطريقة ، حدد الـ Dataset edit box بـ Table1 . هذه الخطوات تربط chart بـ Table قاعدة البيانات . لتحديد أى الحقول يتم عرضها ، اختر الـ Label Name edit box والـ Bar Number . هذه القيم تحدد chart تحسب عدد الزجاجات فى مخزن خمور يحمل بطاقة حقل الـ Name لكل زجاجة . يوضح شكل (١٨-١٢) محرر الـ TeeChart مع الـ Series|Data Source page وهى ممتلئة .

٨- اختر محرر الـ TeeChart لترى الخريطة التامة فى نافذة الـ form ، موضحة المعلومات المأخوذة من قاعدة بيانات الـ WINES . لا يجب عليك تشغيل البرنامج-لإنك قد حددت خاصية الـ Active للـ Table1 بـ True ، فإن البيانات تكون متواجدة داخل Delphi .

لإنهاء البرنامج، يمكنك تغيير عناوين الـ chart واختيار خصائص اخرى.
فى الواقع، هذا سيكون مكاناً جيداً لك لتبدأ واختبار خيارات الـ chart المتنوعة.
توضح قائمة البرنامج [انظر القائمة (١٨-٤)] كيفية برمجة زر الـ Print Preview... كما سبق التوضيح فى هذا الباب.



شكل (١٨-١٢): استخدم محرر الـ TeeChart لربط chart series بالـ Table لقاعدة بيانات، فى هذه الحالة، لقاعدة البيانات WINES

إنشاء تقارير مع الـQuickReport:

يبدو ان طابعات الحاسب اليوم تعمل بشكل جيد، ولكن هذا ليس دائماً.
ليس من زمن بعيد، كانت عملية جعل الطابعة تطبع "أى شئ" تعد إنجازا كبيراً لكثيرين من مستخدمى الحاسب. ولكن الآن، اصبحت طابعات الليزر تحتمل العمل الشاق. ولكن، لكى تجعلها تنطلق كما تريدها مازال امراً يحتاج الى التجربة والخطأ.

ان الـ QuickReport library الخاصة بـ Delphi يمكن ان تبسط متطلبات الطباعة الخاصة بتطبيقك، خاصة اذا كنت تحتاج الى اعداد تقارير مطبوعة من ملف أو بيانات قاعدة بيانات. تأتى مع محرر سهل الاستخدام والذى يبسط تخطيط الصفحة. وكذلك هناك ميزة معاينة الشكل النهائى للطباعة وهى توضح على الشاشة بالضبط كيف ستظهر المطبوعات النهائية، والتى يمكنك استخدامها ايضاً لحفظ تقارير فى ملفات قرص.

Note

ملحوظة: ان التوثيق الكامل للـ QuickReport library متوفر فى دليل تركيب Delphi، فإننى لا اعرف اسم المسار لهذا الدليل، ولكن فى Delphi 4، فإننى لا اعرف اسم المسار لهذا الدليل، ولكن فى Delphi 3، كان المسار، .Delphi 3\Quickrpt). افتح ملف الـ Qrptxman.doc(x تساوى رقم النسخة) باستخدام متصفح الـ Microsoft Word أو الـ Word . وتعتبر online help الكاملة متاحة ايضاً- اختر أى QuickReport component أو بند آخر واضغط F1 لمزيد من المعلومات .

البداية مع التقارير:

ان الـ QuickReport يعتبر مولد banded report . باختصار هذا يعنى انك تنشئ تقريراً بواسطة إدخال واحداً أو أكثر من الـ band objects فى صفحة خالية، وهو object للـ QuickReport . وفى كل bands، تقوم باسقاط عناصر متنوعة تريدها ان تظهر فى التقرير، مثل العناوين، التاريخ والوقت، ارقام الصفحات، حقول تسجيل قاعدة البيانات، إجماليات ملخصة، وعناصر اخرى . عندما تطبع التقرير، تملأ الـ QuickReport bands بالمعلومات وتنسخها حسب الحاجة لإنتاج منتج تام . على سبيل المثال، فى كل صفحة، تقوم الـ QuickReport بصورة تلقائية بزيادة رقم الصفحة، وتملأ حقول التسجيل بمعلومات مأخوذة من tables قاعدة بيانات .

وعندما تعمل مع الـ QuickReport، سوف تكتشف أن هذه الـ bands غاية فى الذكاء . فى اغلب الحالات، يمكنك تصميم تقارير جيدة الشكل بإنشاء bands قليلة وإسقاط objects عليها . إنك تحتاج الى قليل من البرمجة . وميزة معاينة الشكل النهائى للطباعة السهلة الاستخدام، والتى يمكنك استخدامها إما فى نمط تصميم الـ Delphi أو فى تطبيق وقت التشغيل، تجعل من السهل رؤية تخطيط التقرير دون إهدار الورق . عندما يكون كل شئ مضبوط كما تريده، اضغط زر Print الخاص بمعاينة الشكل النهائى للطباعة لترسل النتيجة فى طريقها الى طابعتك . يمكنك ايضاً حفظ التقارير فى ملفات قرص ثم تعيد تحميلها فى نافذة معاينة الشكل النهائى للطباعة التابعة للـQuickReport ليتم طباعتها .

على القرص المدمج: لعرض كيفية استخدام الـ QuickReport components، إتبع الخطوات المذكورة في هذا الفصل والفصول العديدة القادمة. حتى اذا لم يكن لديك طابعة، يمكنك إتباع هذه التعليمات خطوة بخطوة لتتعلم كيفية تصميم تقرير. (والتطبيق التام، Report1، موجود على القرص المدمج في دليل Source\Report1. لاستخدام هذا البرنامج، أو لإنشاءه باتباع الخطوات الموجودة هنا، يجب ان يكون قد قمت بتركيب ملفات عرض قاعدة بيانات Delphi، والمعرفة بالـ Alias name DBDEMOS). ابدأ تطبيقاً جديداً ثم اتبع الخطوات التالية:

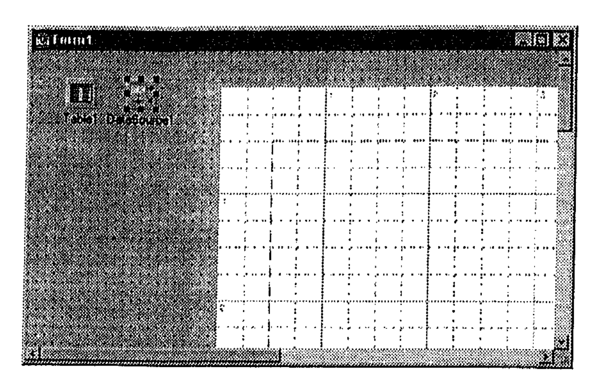
١- اضغط الـ Data Access page tab، اختر component وقم باسقاط على الـ form. اختر الـ DataSource وإسقطه الى جانب الـ Table1 المواضيع بدقة لا تهم- فلا يظهر أي من الـ component في التطبيق التام.

٢- اضغط الـ Table1، قم اضغط الـ Properties tab في نافذة الـ Object Inspector لـ Delphi حدد ثلاث خصائص كما يلي: الـ DatabaseName بـ DBDEMOS، الـ TableName بـ Parts.db، والـ Active بـ True.

ملحوظة: اذا كان الـ DBDEMOS غير مذكور في قائمة اللائحة لخاصية الـ DatabaseName للـ Table1، فإنك تحتاج الى إعادة تركيب Delphi وتبدأ من جديد. أو، يمكنك اختيار أي قاعدة بيانات وجدول آخرين متوفرين على نظامك، ثم حدد أي مجالات متاحة مكان تلك المقترحة هنا. انظر الباب السابع عشر لمزيد من المعلومات عن استخدام components قاعدة البيانات.

٣- اضغط الـ DataSource1، وباستخدام الـ Object Inspector، حدد خاصية الـ DataSet لها بـ Table1.

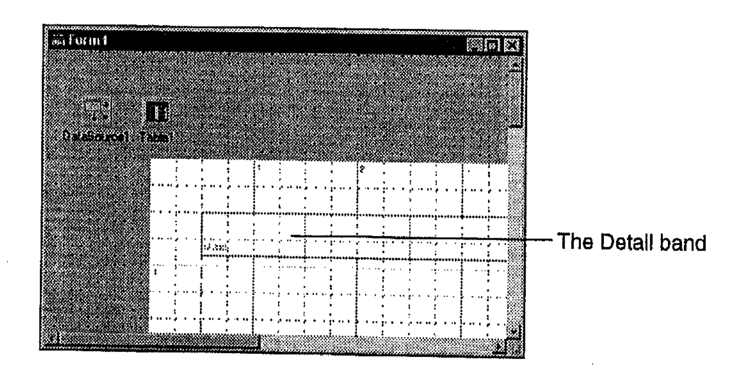
٤- اضغط باب الـ QReport page tab لـ Delphi على الـ VCL palette، واضغط الايقونة الاولى الى اليسار، وهي QuickRep. اضغط داخل نافذة الـ form لإنشاء شكل تقرير خالي، يحمل اسم QuickRep1 حسب النظام الافتراضي. يوضح شكل (١٨-١٣) نافذة الـ form في نمط التصميم حتى الآن. والـ grid وارقام الصف والعمود الخاصة بها والتي تراها على الشاشة هي لتخطيط عناصر التقرير، ولاتظهر في التقرير التام.



شكل (١٨-١٣): شكل تقرير خالى مع components قاعدة البيانات

٥- اضـغـط داخل شكـل التـقـرير الخـالى ، ثـم فى نافـذة الـ Object Inspector ، اضـغـط مرتين عـلامة الزائد الى اليسار من خاصيـة الـBands . هذا يفتح قائمة بالخصائص الفرعية ، كل واحدة لها تحديد True أو False والذى ، عندما يكونTrue ، ينشئ band من هذا النوع فى شكل التقرير .

٦- حـدد الخاصية الفرعية HasDetail بـ True . هذا ينشئ شكل band فى التقرير . اذا نظرت من قرب ، يمكنك ان ترى الكلمة الباهتة "Detail" داخل هذا الـ HasDetail . هذا يعرف نوع الـ HasDetail ، ولكن لا يتم طباعتـه فى التـقـرير التام . ان شكل الـQuickRep1 مع شكل الـ band موضح فى شكل (١٨-١٤) . (انظر على شاشتة عن الـ "Detail" داخل الشريط هذا النص يكون غالباً صغيراً جداً بحيث يصعب رؤيتة فى الشكل المطبوع) .



شكل (١٨-١٤): شكل الـQuickRep1 مع شكل الـ band

٧- تأكد من ان الـQuickRep مازال مختاراً (فهو مذكور فى أعلى قائمة اللائحة الخاصة بالـObject Inspector)، حدد خاصية الـDataSet لها بـ Table1. اصبح الآن شكل التقرير مرتبط بـ components قاعدة البيانات التى يتم اخذ المعلومات منها لإنتاج التقرير التام.

٨- اضغط داخل الـDetailBand1 لاختياره، ولاحظ ان الـ Object Inspector يعرض اسم شكل الـ band، DetailBand1. يقوم الـ QuickReport بنسخ الـDetailband فى التقرير التام لطباعة صفوف من المعلومات.

٩- لتوفير تلك المعلومات، قم بإضافة بعض الـ detail objects، على الـ band . ضع زر QRDBText فى الـ .QReport palette الخاصة بالـ VCL (يوجد العديد من الايقونات فى هذا الفصل من Delphi- اوقف الفأرة على كل زر وانتظر لحظة لترى اسمه هذا يساعدك على إيجاد الـ component الذى تحتاجه). اضغط الـ QRDBText، ثم اضغط داخل الـ Detail band للتقرير. يمكنك سحب شكل النص الناتج (الذى يحمل labeled QRDBText1) الى أى موضع تريده داخل الـ Detail band -أى مكان الى اليسار فى الـ band .

١٠- تأكد من ان الـ QRDBText1 مختاراً، ثم فى نافذة الـ Object Inspector، حدد إثنين من الخصائص كمايلى: الـDataSet لها بـ Table1 والـ DataField بـ PartNo. هذه الخطوات تحدد ان العمود الأول فى التقرير يذكر رقم جزء العنصر. لاحظ ان الـband يوضح اسم الحقل المختار.

١١- اعد الخطوة رقم (٩) لإضافة حقول Table قاعدة بيانات إضافة الى الـ Detail band بالتقرير على سبيل المثال، إختر مرة اخرى الـQRDBText component فى الـpalette، واضغط فى الـ Detail band لإنشاء الـ QRDBText 2. حدد الـDataSet الخاصة بهذا الـobject بـ Table1، ولكن هذه المرة، حدد الـDataField بـ Description اضف QRDBText components إضافية لحقول الـCost، والـListPrice، والـOnHand، والـ OnOrder. اعد تحديد حجم نافذة الـform على حسب الحاجة لعرض مزيد من التقرير المتطور. يوضح شكل (١٨-١٥) الـDetail band التام.



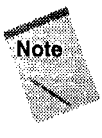
فكرة: إعتماداً على نوع الحقل الذى تقوم بطباعته فى العمود، قد تريد ان تحدد خاصية الـAlignment له بشئ مختلف. على سبيل المثال، ان عمود الـ Description فى التقرير يجب ان يحدد الـ Alignment بـ taLeftJustify. والحقول الرقمية الاخرى سوف تبدو افضل اذا تم تحديدها بـ taLeftJustify يجب عليك ان تعمل بهذه أو تلك الخصائص، وقد تحتاج ان تعدل مواضع الاعمدة، لتحصل على النتائج النهائية كما تريدها.

شكل (١٨-١٥) الـ Detail band التام. كل QRDBText يمثل حقل Table قاعدة بيانات يجب طباعتة فى التقرير التام.

١٢- احفظ التطبيق الآن. إنك تستمر فى تطوير هذا التقرير فى الفصول التالية. استخدم الوحدة الافتراضية واسماء المشروع الافتراضية، واحفظهم فى أى دليل مؤقت.

تحذير: يعتبر band التقرير object containers . هذا يعنى انه، اذا قمت بحذف band، فإنك ايضاً تحذف الـobjects الموجودة فى الـ band. والأهم من ذلك. التابعة لـ QuickRep بـ False، أو يكون عليك ان تعيد ادخال كل الـ objects التى ادخلتها بعناية فى هذا الـ band. للسلامة، بعد إنشاء كل band وإدخال الـ band فيه، احفظ مشروعك.

تنتج الخطوات، اضغط داخل الـ Detail band). اختر امر الـ Preview من القائمة لرؤية التقرير كما تم تجميعه حتى الآن. يوضح شكل (١٨-١٦) النتائج.



ملحوظة: بالرغم من انك تجرب اوامر معاينة الشكل النهائى والطباعة الخاصة بالـ QuickReport داخل Delphi، يرى المستخدمون النهائيون

لتطبيقك نفس نافذة dialog معاينة الشكل النهائى . مؤخراً فى هذا الباب ، سأشرح كيفية توفير هذه الميزة فى تطبيقاتك التامة .

900	Dive kayak	$1,356.75	$3,999.95	24	16
912	Underwater Diver Vehicle	$504.00	$1,680.00	5	3
1313	Regulator System	$117.50	$250.00	165	216
1314	Second Stage Regulator	$124.10	$365.00	98	88
1316	Regulator System	$119.35	$341.00	75	70
1320	Second Stage Regulator	$73.53	$171.00	37	35
1328	Regulator System	$154.80	$430.00	166	100
1330	Alternate Inflation Regulator	$85.80	$260.00	47	43
1364	Second Stage Regulator	$99.90	$270.00	128	135
1390	First Stage Regulator	$64.60	$170.00	146	140
1946	Second Stage Regulator	$95.79	$309.00	13	10
1986	Depth/Pressure Gauge Console	$73.32	$188.00	25	24
2314	Electronic Console	$120.90	$390.00	13	12
2341	Depth/Pressure Gauge	$48.30	$105.00	226	225

شكل (١٨-١٦): امر الـ Preview الخاص بالـ QuickReport يعرض التقرير كما سيظهر على الصفحة المطبوعة

جرب الازرار الموجودة على صفحة معاينة الشكل النهائى للطباعة . اضغط Zoom لعرض الصفحة كاملة ، اضغط Zoom to 100 لتحصل على تمثيل واضح على الشاشة للتقرير المطبوع (هذا الخيار مفيد لوضع الـ objects بعناية قبل الطباعة) . اضغط ازرار الاسهم لترى صفحات التقرير . اضغط Printer setup لتعد خيارات طابعتك ؛ اضغط Print للطباعة (تبدأ الطباعة على الفور- فهى لا تظهر خصائص طباعة ، لذا تأكد من اختيار Printer setup قبل ذلك) . وهناك زرين آخرين وهما save و Open ، يحفظان التقرير الحالى فى ملف قرص ويعيدا تحميل القارير التى تم حفظها فيما سبق . اضغط زر الـ Close لتعود الى صفحة chart form .

طباعة رؤوس العمود:

حتى الآن، يبدو تقريراً مقبولاً، ولكن الاعمدة غير معرفة، واغلب المعلومات فى الإخراج النهائى ستكون بلا معنى [انظر شكل (١٨-١٦)]. ان كل عمود يحتاج الى الـ label، وافضل ما يكون طباعتها باستخدام بنطاً مختلفاً ليكون رأس العمود بارزاً.

ورأس العمود هو مجموعة من الـ QRLabel، والتى تستطيع ان تظهر أى نص تريده. وهذه الـ objects تشبه الـ label الخاصة بـ Delphi، ولكن تم تصميمها لتدخل فى الـ QuickRep object bands. ولان، فى هذه الحالة، يجب طباعة رأس العمود فقط على قمة كل صفحة فانت تحتاج ان تنشئ band تقرير مختلف ليحمله. اذا وضعت labels فى Detail band، فستتكرر فى كل سطر جديد من المخرجات (قد يكون هذا مفيداً فى بعض الاحوال، ولكنه ليس التأثير المطلوب هنا).

على القرص المدمج: اذا كنت مازالت ترى نافذة معاينة إغلقها لتعود الى Delphi، ثم اتبع الخطوات التالية لإضافة رؤوس عمود الى التقرير (مرة اخرى، تطبيق التقرير التام موجود على القرص المدمج فى دليل Source\Report1):

١- اختر شكل QuickRep1 (يجب ان يظهر اسمه فى نافذة الـ Object Inspector). اضغط مرتين علامة الزائد الواقعة بعد خاصية الـ Bands، وحدد خاصية الـ HasColumnHeader بـ True. هذا يضيف آخر Band للـ QuickRep1، يحمل الـ label الباهتة ColumnHeader (هذا النص لا يظهر فى التقرير التام). ويوضح شكل (١٨-١٧) التقرير مع الـ Column Header band. الجديد.

٢- اختر QRLabel من الـ VCL QReport palette، واضغط داخل الـ Column Header band لإدخال label تحمل اسم QRLabel1 حسب النظام الافتراضى. اسحب الـ QRLabel1 الى أعلى العمود الذى يحتاج، ثم استخدم الـObject Inspector لإدخال كتابة أى نص تريده- لايجب ان يتناسب مع اسم حقل الرأس. يوضح شكل (١٨-١٨) الـ Column Header band التام مع الـQRLabel أعلى كل عمود.

Column Header band Set to True

شكل (١٨-١٧): حدد الـ HasColumnHeader بـ True لإضافة Column Header band، والذى يطبع labels عمود على كل صفحة من التقرير التام

٣- من الافضل دوماً ان تطبع رؤوس الاعمدة باستخدام fonts مختلفة . يمكنك اختيار fonts لكل QRLabel الاخرى باستخدام خصائص الـ Font لهم . ولكن، لكى تغير الـ font للـ band باكمله، اختر band الـColumnHeader قائمة لائحة نافذة الـObject Inspector توضح الـ (ColumnHeaderBand1)، ثم اضغط الزر البيضاوى الواقع بعد خاصية الـ الـFont . فى dialog اختيار الـ font الناتج، اختر Bold . يمكنك ايضاً ان تختار الوان المخرجات، اسماء عائلة الـ font، احجام النقاط، وان تصنع التغييرات الاخرى فى الـ font المختار وخصائصة .

شكل (١٨-١٨): الـ Column Header band التامة لكل عمود مع الـQRLabel

٤- عندما تنتهى من وضع labeling الاعمدة، اضغط داخل الـQuickRep1 (لاتضغط داخل أى من الـ two bands)، واختر امر الـ Preview من القائمة المحلية. يمكنك عندئذ معاينة الشكل النهائى وطباعة التقرير بكل اعمدة حاملة لـ labeled.

٥- احفظ التطبيق الآن. إنك تستمر فى تطوير هذا التقرير فى الفصول القادمة.

Tip

فكرة: تأخذ كل الـ objects خصائص الـ font من الحاوية الأم لهم. يستخدم الـ bands الـ font للتقرير، والـ objects الموجودة فى الـ bands تستخدم band fonts، الى آخر ذلك. لتغيير الـ font للتقرير باكمله، اختر الـ QuickRep وحدد خاصية الـ Font له. يمكنك عندئذ تعديل خصائص الـ Font للـ bands المنفردة والـobjects المحتواه فيها.

طباعة معلومات النظام:

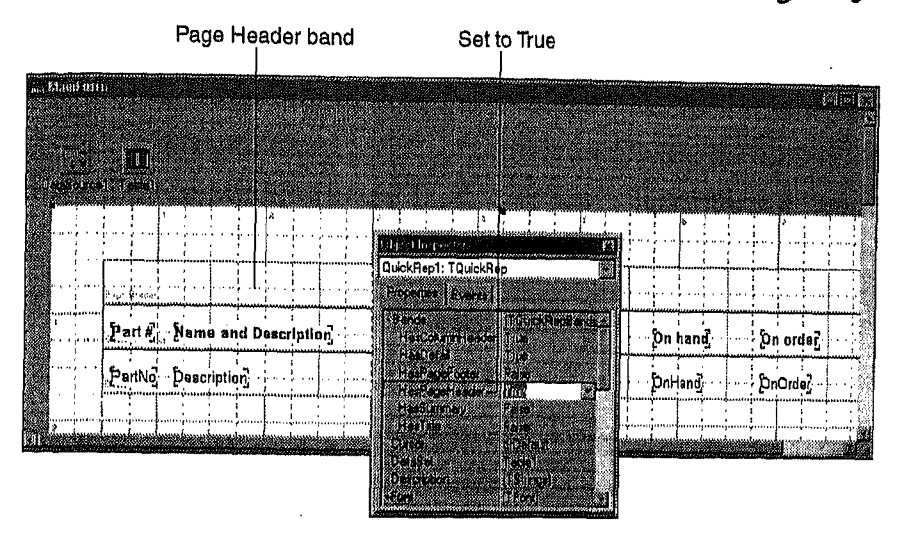
إنك غالباً تريد ان تضيف معلومات نظام لصفحات التقرير. وهذه المعلومة قد تعرض عنوان تقرير، التاريخ والوقت الحاليين، ورقم صفحة. كما فعلت مع labels العمود والحقول الخاصة بالتقرير، لإضافة معلومات نظام، فإنك تنشئ نوع آخر من الـ band تسقط فيه QuickReport .

اغلق نافذة معاينة الشكل النهائى الخاصة بالـQuickReport اذا كانت مفتوحة، وإرجع الىDelphi . ثم، إتبع الخطوات التالية لإضافة عنوان، تاريخ ووقت، ورقم صفحة للتقرير:

١- اختر QuickRep1 ، اضغط مرتين علامة الزائد الواقعة الى اليسار من خاصية الـ Bands، وحدد خاصية الـ HasColumnHeader بـ True. هذا يضيف آخر Band ثالث الى التقرير، كما هو الحال مع الـ band الأخرى، يظهر النص الـ Page Header فقط فى وقت التصميم، وليس فى المخرجات المطبوعة. (قد يكون عليك ان تنظر الى شاشتك لترى هذا النص).

٢- ان الـ objects الموجودة فى الـ Page Header band تكون غالباً QRLabel أو QRSysData objects . لإضافة عنوان الى أعلى الصفحة، اختر

QRLabel من الـ QReport VCL palette، واضـغط داخل الـ Page Header band لإنشاء الـ QRLabel1 . حدد خاصية الـ Caption له، واختر حجم font، نمط، ولون باستـخدام خاصيـة الـ Font . (لقـد استخـدمت Arial ، ١٤ نقطة، Bold Italic، لون الازرق الغامق (Navy blue). بعـد كل هذا، هذا التقرير هو لمخزن محل).



شكل (١٨-١٩): إنشئ Page Header band لطباعة عناوين، تواريخ واوقات، ارقام صفحات، معلومات اخرى فى أعلى كل صفحة

٣- اختـر QRSysData، وقم باسـقـاط اثنين منه فى الـPage Header band . كل واحـد من هذين الـ two components يمكن ان يعـرض انواع مـختلفة من مـعلومـات النظام الثـابتـة. اختـر كل object، وحـدد خـاصـيـة الـData بنوع المعلومـــات الذى تريده-فى هذه الحـــالة، لقــــد حــــددت الـobject الاول بـ qrsDateTime والثانى بـ qrs PageNumber . ضع الـ two components اينما تريد. يوضح شكل (١٨-٢٠) التقرير مع الـPage Header band التام.

٤- احـفظ التـقرير الآن. إنك تستـمـر فى تطوير هذا التـقـرير فى الـفـصـول القادمة.

كما فعلت فى السابق، اضـغط يميناً داخل الـQuickRep1 (وليس داخل الـ bans)، وثم اختر امر الـ Preview . كـمـا يوضح شكـل (١٨-٢١)، يبـدأ التـقرير

التام فى الظهور بشكل طيب ، ولكن بعض البنود يمكن اضافتها لتنمية بعض الشيء .

QRLabel object

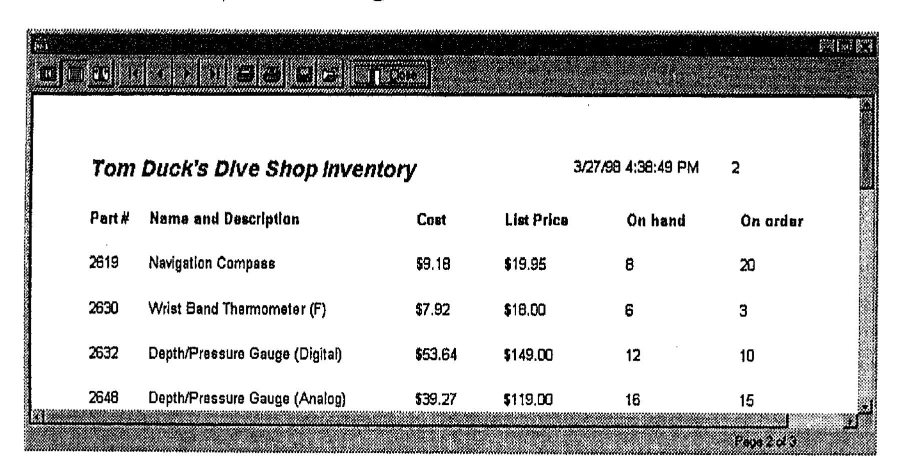
QRSysData objects

Tom Duck's Dive Shop Inventory [Date/Time] [Page#]

[Part #] [Name and Description] [Cost] [List Price] [On hand] [On order]

[PartNo] [Description] [Cost] [ListPrice] [OnHand] [OnOrder]

شكل (١٨-٢٠): الـ Page Header band الكامل مع عنوان QRLabel و QRSysData objects موضحة التاريخ والوقت، ورقم الصفحة



Tom Duck's Dive Shop Inventory 3/27/98 4:38:49 PM 2

Part #	Name and Description	Cost	List Price	On hand	On order
2619	Navigation Compass	$9.18	$19.95	8	20
2630	Wrist Band Thermometer (F)	$7.92	$18.00	6	3
2632	Depth/Pressure Gauge (Digital)	$53.64	$149.00	12	10
2648	Depth/Pressure Gauge (Analog)	$39.27	$119.00	16	15

شكل (١٨-٢١): التقرير الذى أوشك على الاتمام مع عنوان الصفحة، تاريخ ووقت، رقم صفحة، رؤوس اعمدة، ومعلومات قاعدة بيانات



فكرة: بدلاً من استخدام الـPage Header band لطباعة عنوان التقرير ، يمكنك تحديد الخاصية الفرعية HasTitle للـ Bands بـ True . يمكنك عندئذ إضافة title label للـ Title Band–هذا الـ Band يتم طباعة على الصفحة الأولى فقط .

لطباعة معلومات فى اسفل كل صفحة ، حدد الخاصية الفرعية HasFooter للـ Bands التابعة للـ QuickRep بـ True . استخدام band الناتج تماماً مثل الـ Header Band الموضح فى هذا الفصل . بالتأكيد ، يتم طباعة الـ object التى تدخلها فى هذا الـ band فى اسفل كل صفحة بدلاً من اعلاها .

تلخيص الاعمدة:

لجمع القيم فى عمود رقمى ولتوفير معلومات ملخصة اخرى ، فإنك تحتاج الى band واحد اضافى واثنين من الـ components لهذا التقرير على سبيل المثال ، التقرير المتطور [راجع شكل (١٨-٢١)] قد يجمع القيم فى اعمدة On Hand أو On Order . ولأغراض الايضاح ، قمت ايضاً باظهار مجموع لعمود الـ List Price ، بالرغم من ان هذا لا يبدو مفيداً فى هذه الحالة . ولكن ، القيمة الإجمالية لمخزن المحل ستكون مفيد للغاية ، لذا صممت هذا فى التقرير النهائى .

لإضافة band تلخيص الى التقرير ، اغلق نافذة dialog معاينة الشكل النهائى للـ QuickReport اذا كانت مفتوحة ، وراجع الى Delphi . ثم اتبع الخطوات الآلية :

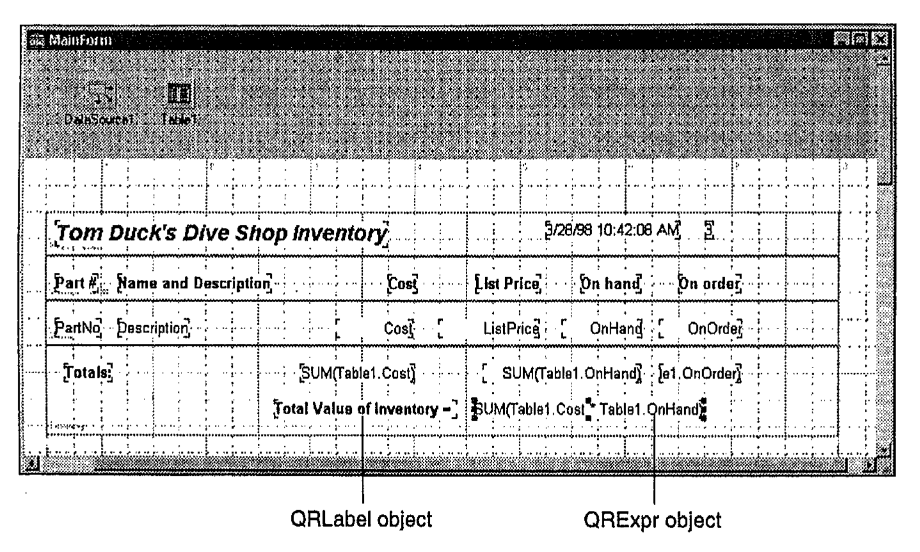
١- اختر QuickRep1 ، وفى الـ Object Inspector ، اضغط مرتين خاصية الـ Bands ، وحدد خاصية الـ HasSummary بـ True . يوضح شكل (١٨-٢٢) شكل التقرير ، النامى مع band الجديد . كما هو الحال مع الـ bands الأخرى ، يظهر النص الـ Summary فى Delphi فقط ، وليس فى المخرجات المطبوعة .

٢- يظهر الـ Summary band فى اسفل chart بعد السطر الاخير من البيانات . يمكنك اضافة QuickReport متنوعة الى هذا الـ band . فى هذا الحالة ، نصف تحتاج الى نوعين QRLabel objects لتعريف قيم التلخيص و QRExpr objects لأداء الحسابات . (تحمل ايقونة الـ QRExpr لـ labeled E = mc#2) . صنع اثنين من الـ QRLabel objects على الـ Summary band واربعة من الـ QRExpr objects . اضغط واسحب الخط الخارجى للـ Summary band لتكبيره ، ورتب الـ objects كما هو موضح فى شكل (١٨-٢٣) . حدد خصائص الـ QRLabel Caption كما هو موضح فى الشكل ، واستخدم خاصية الـ Font لاختيار لون للمخرجات .

Set to True

Summary band

شكل (١٨-٢٢) : اضف Summary band الى التقرير لطباعة اجماليت اعمدة ومعلومات ملخصة اخرى فى اسفل التقرير



شكل (١٨-٢٣): الـ Summary band التام مع الـ QRLabel objects والـ QRExpr

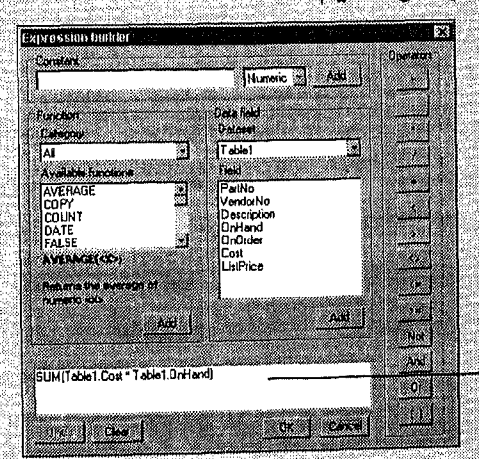
٣- لحساب قيم الـ Summary، اختر كل QRExpr ، وحدد كل خاصية Expression كما يلى . ويتم عرضها فى الـ form المصممة وفى التقرير التام ، يتم استبدالها بالقيم المحسوبة . لاحظ انه للإشارة الى حقل فى Table قاعدة البيانات ،

يشير التعبير الى الـTable1 الخاص الـ form باستخدام نظام النقاط زائد اسم الحقل (Cost، OnHand، الى آخره). وهذه ليست برمجة Pascal- يتم تقييم نفس التعبير بواسطة QuickReport :

```
Sum(Table1.Cost)
Sum(Table1.OnHand)
Sum(Table1.OnOrder)
Sum(Table1.Cost * Table1.OnHand)
```

منشئ تعبير الـ QuickReport Expression Builder

بدلاً من كتابة التعبيرات فى خصائص الـ QRExpr Expression، يمكنك ان تضغط الزر البيضوى لهذه الخاصية لفتح الـ QuickReport Expression Builder (أو منشئ تعبير الـ QuickReport). انظر شكل (١٨-٢٤). استخدام المحرر، اختر Function، واضغط Add فى الـ panel اليسرى. على سبيل المثال، اختر SUM. بعد ذلك، اختر Dataset object (هناك واحداً فقط فى هذا المثال، وهو Table1)، واختر Field مثل Cost. اضغط Add فى الـ panel اليسرى لتضيف هذا الى التعبير. اختيارياً، اضغط Operator فى الـ panel اليمنى البعيدة، ثم اختر Field آخر مثل OnHand. اخيراً، اضغط OK لاتمام التعبير، الذى يكون موضحاً فى النافذة السفلى. عندما ترضى عن هذا، اضغط OK مرة اخرى لتدخل التعبير فى خاصية الـ QRExpr Expression وتعود الى تصميم . الـform.



شكل (١٨-٢٤): بدلاً من إدخال خصائص الـ QRExpr Expression بالكتابة، يمكنك استخدام الـ QuickReport Expression Builder لإنشاء عبارات

٤- بالإضافة الى تحديد خصائص الـ QRExpr Expression، فى الغالب تريد ان تدخل خاصية Mask لتصوير القيمة المطبوعة. قد تريد ايضاً ان تفعل نفس الشئ للـ QRDBText فى الـ Detail band–ان الـ Masks تعطيك تحكماً افضل فى محاذاة الاعمدة، وتمكنك من تحديد عدد الاماكن العشرية فى القيم الرقمية. فيما يلى خصائص الـ mask المستخدمة فى التقرير ونماذج من نتائجها:

#,###.00	2,609.20
$#,###,###.00	$24,891.05
####	43

Masks

فى الـ mask، علامة (#) تمثل أى خانة رقمية، ولكن يتم طباعتها فقط اذا كانت القيمة لها رقم فى هذا الموضع. والصفر يمثل أى رقم، ويتم طباعته دائماً حتى اذا لم يكن بالقيمة رقم فى هذا الموضع. والفصلات تشير الى الفاصل الذى تريد استخدامة فى القيم الكبيرة.

ان نوع الـ mask يختلف اعماداً على نوع حقل البيانات. ان الـ TDateField، والـTDateTimeField، والـ TTimeField objects يتم تهيئتها باستدعاء الـ DateTimeToStr functionالخاصة بـ Delphi. ولهذا الـ components، اذا لم يتم ادخال خاصية الـ Mask ، تتطابق المخرجات مع المواصفات الموجودة فى ملف بدء الـWindows، وهو Win.ini، فى القطاع [الدولى].

ان الـ TBCDField، والـ TCurrencyField، والـ TFloatField يتم تحديد الـformat لها باستدعاء الـ FloatToTextFmt function التابعة لـ Delphi. لهذه الـ components، اذا لم يتم إدخال Mask أو Display form ، يتم تحديد format القيمة باستخدام خاصية الـ Currency للمجال.

٥- بعد تحديد خصائص الـ Expression والـ Mask فى الـ QRExpr الخاصة بك، اضغط يميناً الـQuickRep1 (لاتضغط داخل الـ band)، واختر امر الـPreview من القائمة المحلية. يوضح شكل (١٨-٢٥) معلومات التلخيص للتقرير التام مع إجمالى العمود وتقرير عن قيمة المخزن الإجمالى للمحل.

٦- احفظ التطبيق الآن. سوف تستمر فى تطوير هذا التقرير فى الفصول القادمة.

بيانات التقرير Sorting:

ان الـ sort تعـتـبـر أحـد عـمليـات قـاعـدة بيـانـات ، وليس QuickReport function . لترتيب التقرير ، فإنك تختار index لـ table قاعدة البيانات اذا لم يوجد index للحقل ، استخدم الـ Database Desktop لإنشاء واحداً .

Tip **فكرة:** ولكنك قـد تتسـاءل ، مـاذا لـو كنت لاتستطيع ان تنشئ index جديداً لان Database Desktop قـاعـدة بيـانات للقـراءة فقط ، أو انهـا متاحة فقط عبر Server بعيد؟ فى مثل هذه الحالات ، اقترح عليك ان تطبع التقرير على ملف نص قرص ، ثم قم بترتيب سطور النص . اذا كـان لديك متطلبات ترتيب خاصة ، يمكنك قراءة الـ code لتقوم بالترتيب ، أو اذا كنت تريد ترتيباً ابجدياً أو رقمياً ، فيمكنك ان تفعل كما فعلت انا . قم بتحميل النص فى برنامج معالج كلمات خاص بك واستخدم امر الـ Sort للبرنامج . يمكنك عندئذ ان تطبع ملف النص المرتب . وهذا عادة يكون اسهل بكثير من تجربة ان ترتب قاعدة بيانات حقيقية .

لترتيب تقرير المحل الخاص بنا بوصف العنصر ، اغلق نافذة مراجعة الشكل النهائى للـ QuickReport اذا كـانت مـفـتـوحـة وعـد الى Delphi . اتبع هذه الخطوات :

١ - اختـر Table1 . فى الـ Object Inspector (اضـغط F11 اذا لم تكن مرئية) ، اضغط السهم المشير الى اسفل الواقع يعد خاصية الـ IndexName . هذا يعرض indexes الحقل المتاحة . اختر ByDescription [انظر شكل (١٨-٢٥)] .

٢- اضغط يميناً الـ QuickRep1 (ولكن ليس داخل الـ band) واختر أمر الـ Preview لترى وتطبع التقرير التام .

٣- اضغط التطبيق الآن . إنك تحتاجه مرة أخيرة فى الفصل التالى .

طباعة التقارير وقت التشغيل:

حتى الآن ، لقد قمت بتطوير ، معاينة الشكل النهائى وطباعة تقارير من داخل Delphi تماماً . اذا كنت تحتاج الى إنشاء تقارير قاعدة بيانات لنفسك ، فهذا مقبول تماماً- فلا يجب عليك أن تكتب أى code أو تتم التطبيق . ولكن ، قد تحتاج أن توفر

إمكانات توليد وتقرير للمستخدمين النهائيين لتطبيقك، وتعطيهم كل نفس مميزات معاينة الشكل النهائي والطباعة المقدمة في هذا الباب.

Select Table — Choose index name

شكل (١٨-٢٥): الترتيب هو عملية قاعدة البيانات وليس التقرير. لترتيب تقرير، اختر الـ Table واختر الـ index المذكور في خاصية الـ IndexName

On the CD-ROM

على القرص المدمج: اتبع الخطوات التالية لإتمام تطبيق الـ Report1 ولإضافة Buttons لمعاينة الشكل النهائي وطباعة التقرير التام. يوجد البرنامج التام على القرص المدمج في دليل Source\Report1. يمكنك تحميل ملف مشروع الـ Report1.dpr في Delphi، أو اذا كنت متابعاً، أغلق نافذة معاينة الشكل النهائي للـ QuickReport اذا كانت مفتوحة وإرجع الى Delphi، وإتبع هذه الخطوات:

١- أضف إثنين من BitBtn من الـ Additional VCL الى الـ form، وحدد خصائص الـ Caption لها بـ...Preview ويمكنك إضافة Print BitBtn ثالث وتحدد خاصية الـ Kind له بـ bkClose لتوفر طريقة سريعة للمستخدمين للخروج من التطبيق).

٢- إختيارياً، قم بتحميل bitmaps في خصائص الـ Glyph لكل زر. لقد استخدمت صور الـ Report.bmp والـ Print.bmp المتوفرة مع Delphi في دليل التركيب، Images\Buttons.

٣- اضـغط مـرتين الـ BitBtn1 لإنشـاء OnClick لزر الـ ...Preview . إمـلأ الـ procedure باستخدام القائمة (١٨-٥) كمرشد لك . اضغط مرتين الـ BitBtn2 لإنشـاء OnClick ثانى لزر الـ Print . إمـلأ هذا الـ procedure أيضـاً باسـتـخـدام القائمة (١٨-٥) كمرشد لك .

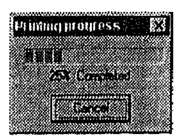
٤- اضـغط F9 للـ compile وتشـغيل البـرنامج . يوضـح شكل (١٨-٢٦) العـرض التـام للبـرنامج . اضـغط زر الـ ...Preview لإظهـار نافـذة الـ Preview للـ QuickReport . اضغط Print لطباعة التقرير التام- تبدأ الطباعة بمجرد أن تضغط هذا الزر . اذا جـربت هذا، فـإنك ترى progress indicator على الشـاشـة [شكل (١٨-٢٧) أثناء إنشـاء المطبـوعات وإرسـالهـا الى الطابعـة . إنك ترى أيضـاً ومضـة بيانات فى الـ components المختلفة للتقرير .

شكل (١٨-٢٦): العرض التام لتطبيق الـ Report1 مع أزرار الـ ...Preview والـ Print

القائمة (١٨-٥): Report1\Main.pas (OnClick للـ BitBtn)

```
{ Respond to click of the Preview... button }
procedure TMainForm.BitBtn1Click(Sender: TObject);
begin
  QuickRep1.Preview;
end;
```

```
{ Respond to click of the Print button }
procedure TMainForm.BitBtn2Click(Sender: TObject);
begin
  QuickRep1.Print;
end;
```



شكل (١٨-٢٧): أثناء الطباعة، يرى المستخدمون الـ progress indicator

افكار للمستخدم الخبير

- أثناء استخدام الـ TeeChart editor، اضغط الـChart والـ General page tabs، ثم اضغط زر الـ ...Print Preview لمعاينة الشكل النهائى وطباعة الـ chart أو، يمكنك ان تضغط Export لتكتب الـ charts للملفات الحجم الكبير بالقرص أو ملفات الـ bitmap وهذه الازرار اسهل بكثير فى الاستخدام اثناء تصميم chart عن الخروج من محرر الـ TeeChart والضغط يميناً داخل الـ chart لمعاينة شكلها النهائى.
- لإضافة chart axis labels ثابتة، كتلك المستخدمة فى تطبيق الـ MayTemp لهذا الباب، افتح محرر الـ TeeChart ، اختر الـ Chart page tab، اضغط الصفحة الفرعية Axis. داخل هذه الصفحة، هناك series اخرى من page tabs. اختر Scales وابطل الـ Automatic check box. بعد ذلك اضغط كل واحد من زرى الـ ...Change وادخل الحد الادنى والحد الأعلى من القيم لتظهر على المحور المختار على اليسار. (تأكد من ان الـ check box الظاهر تم تشغيله، أو ان تغييراتك لن تظهر على الـ chart التام). اذا اردت ان تحدد الحد الادنى أو الحد الأقصى فقط، اضغط زر الـ Change للقيمة، وقم بتشغيل الـ Automatic check box للقيمة الأخرى.

- اذا جربت الفكرة السابقة ، لا تفحص الـ Automatic check box مرة اخرى ، أو سيكون عليك ان تبطله وتعيد إدخال قيم قياس الحد الادنى والحد الاقصى الخاصة بك .

- لمنع objects التقرير من ان تنطبع فوق بعضها ، حدد خصائص الـ AutoSize لها بـ True . كذلك اضغط واسحب الـ objects الفردية (QRDBText ، QRExpr ، QRLabel ، والاخرى) لتحديد حجمها . استخدم خاصية الـ Mask لتصور formats المخرجات .

- من الممكن ان تحصل على Report object ممتد لتحصل على الـ band الخاص به ممتد رأسياً اذا لم يكن هناك مساحة كافية لطباعة نص لـ object . لتفعل هذا ، حدد الـ AutoStretch بـ True . ولكن ، يجب ان يكون component واحد فقط لكل band هو الذى له هذه الخاصية محددة بـ True - اذا تم تحديد أكثر من object واحد للامتداد بصورة تلقائية ، ستكون النتيجة النهائية غير متوقعة (وغير مقروءة طبعاً) .

- ان التطبيق Report1 فى هذا الباب له اوامر ...Preview و Print ولكن ، لان نافذة معاينة الشكل النهائى للـ QuickReport لها زر الـ Print الخاص بها ، قد لا يكون ضرورياً ان تضيف امر الـ Print لتطبيقاتك - ان استدعاء الـ QuickRep Preview method يوفر كل خدمات الرؤية والطباعة .

- لطباعة تقارير فى الخلفية ، فبدلاً من استدعاء الـ QuickRep Print method ، استدع PrintBackground . هذا يمكن المستخدمين من الاستمرار فى استخدام البرنامج أثناء طباعة التقارير الطويلة . يجب أن يكون لديك نسخة ذات 32-bit من Delphi (Version 2 أو ما بعدها) لتستخدم الـ PrintBackground .

- لإضافة chart الى تقرير مطبوع ، اختر الـ QRChart من الـ QReport VCL ، وأدخل هذا الـ object فى band تقرير . بالرغم من أن الـ QRChart موجود على الـ QRChart ، فإنه TeeChart ، وهو موثق فى ملفات الـ TeeChart online .

المشروعات التى يمكنك تجربتها

١٨-١ : قم بتعديل مشروع الـ MayTemp لتحميل ملفات أخرى عن بيانات درجة الحرارة . على سبيل المثال ، قد يستخدم برنامجك أمر الـ File|Open ، أو زراً ، لعرض dialog ملف مفتوح . اجعل المستخدمين قادرين على فتح ملف درجة حرارة وأقرأ البيانات الموجودة فى chart السطر للبرنامج .

١٨-٢ : أضف أمر قائمة أو زر print-preview لواحد من الـ chart النموذجية بهذا الباب وهى MemInfo أو MayTemp .

١٨-٣ : أضف أمر الى التطبيق WinesChart لضغط الـ chart فى bitmap . اعرض dialog حفظ ملف لتمكن المستخدم من اختيار دليل واسم ملف للمخرجات .

١٨-٤ : قم بإنشاء تقارير مطبوعة للتطبيقات الموجودة بهذا الباب وهى MayTemp و MemInfo و WinesChart .

ملخص:

- توفر الـ TeeChart library إمكانات واسعة لصنع الـ charting . تتوفر نسخة كاملة للـ TeeChart library مع Delphi . والنسخة المتخصصة مع الـ source code الكاملة والـ components الإضافية متاحة لدى موزع المنتج ؛ وهو teeMach .
- اختر chart series لإنشاء أنواع مختلفة من الـ pie charts و bar charts للـ الـ chart object أن يكون له series objects واحد أو أكثر ، والذى قد يكون من نفس النوع أو مختلف .
- اضغط يميناً داخل الـ Chart واختر الـ ...Print Preview لطباعة ومعاينة للـ charts داخل Delphi .
- لمعاينة الشكل النهائى للـ chart فى وقت التشغيل ، والذى يعطى المستخدمين أيضاً القدرة على طباعة الـ chart ، أضف TeePrevi لتقرير الـ

uses للـ unit ، واستدع الـ ChartPreview procedure كما تم التوضيح فى التطبيق WinesChart لهذا الباب . (يوجد هذا التطبيق على القرص المدمج فى دليل Source\WinesChart) .

● توفر الـ QuickReport library إمكانات توليد واسعة للتقرير . تتوفر نسخة كاملة من الـ QuickReport library مع Delphi . تتوفر نسخة متخصصة مع الـ source code التـامـة و components إضـافـيـة مع مـوزع المنتج . وهو QuSoft .

● يعتبر الـ QuickReport «مولد banded report» لإنشـاء تقرير ، قم بإضـافة QuickRep object على form . استخدم خـاصية الـ Bands لهذا الـ objects لإنشاء Bands . قم بإضافة objects فردية على هذه لبرمجة التقرير . لا تحتاج الا لقليل من البرمجة لإنشاء حتى اطول انواع التقارير وأكثرها توسعاً .

فى الجزء التالى والاخير من هذا الكتاب ، سوف تعرف موضوعات متقدمة فى Delphi وبرمجة الـ Object Pascal ، بما فى ذلك الـ exception وانشاء الـ component وتلك الموضوعات مثل كتابة الـ DLLs ، استخدام ادوات خط اوامر Delph ، كتابة event handlers على مستوى التطبيق ، كتابة event handlers على مستوى التطبيق ، ومايزيد عن ذلك .

*****

الجزء الرابع
تقنيات متقدمة

محتويات هذا الجزء:

- الباب التاسع عشر: التعامل مع الـexceptions.
- الباب العشرون: انشاء الـCustom Components.
- الباب الحادى والعشرون: تطوير مهارات الـDelphi الخاصة بك.

بعد برمجة التطبيق ، لايحاول الكثير من المستخدمين التعرف على المزيد من الاشياء . ولكن اغلب المطورين الناجحين يأخذون بعض الوقت لإضافة اللمسات النهائية التى تستطيع ان تحول التطبيق الجيد الى عمل رائع من البرمجيات .

إن الابواب الموجودة فى الجزء الرابع تتناول تقنيات برمجة Delphi المتقدمة . سوف تجد تفسيرات عن كيفية التعامل مع الاخطاء عن طريق الـ exception و عن كيفية إنشاء Custom Components و ActiveX control ، وحول كيفية تطوير مهارات Delphi الخاصة بك مع افكار وتقنيات برمجة ربما فى ذلك مقدمة عن كتابة تطبيقات الانترنت .



الباب التاسع عشر
التعامل مع الـ exceptions

محتويات هذا الباب:

- معلومات حول الـ exceptions.
- التعامل مع الـ exceptions.
- إنشاء الـ exception classes.
- تقنيات أخرى للـ exceptions.

ان التعامل مع الاخطاء هو أحد الضروريات السيئة فى حياة مطور البرمجيات. فلا مهرب من هذه المهمة- اذا فشلت تطبيقاتك نتيجة لظرف خاطى، فسيكون برنامجك منتشر مثل الفراء فى المناطق الاستوائية اثناء موجه حارة. يتعامل التطبيق القوى مع كل الظروف الممكنة والتى قد تجعل البرنامج يتوقف أو ينتج نتائج غير صحيحة.

ومثل هذه الظروف تسمى exceptions والتعامل معها هو موضوع هذا الباب. يوفر الـ Object Pascal نظام يعمل مع الـ exception غاية فى التعقيد، وكل تطبيق Delphi له exception handlers حسب النظام الافتراضى له يمكن ان يعرضوا رسائل الخطأ ويمنعوا البرنامج من التوقف الغير متوقع اذا حدث exception. وبدمج exception-handling إضافية، فيمكنك ان تحميك استجابات تطبيقك ضد الظروف الغير متوقعة، وعندما تحدث مشكلة، تقوم باعمال المسح اللازمة مثل تحرير الذاكرة المخصصة، حفظ البيانات الاساسية، واغلاق الملفات.

معلومات حول الـ Exceptions:

يقوم المطورن ذوى الخبرة بدمج معالجة الاخطاء فى مشروعاتهم اثناء دورة التطوير . ان معالجة الـ exception فى الـ Object Pascal تعتبر تقنية مختصة بالـ object والتى تقلل كتابة هذا الـ code ان الـ exception تسهل تطوير الـ code و التعامل مع الخطأ معاً، وكما ستعرف فى هذا الباب ان الـexception handlers لا يحاولون الوقوف فى طريق حسابات البرنامج كما تفعل التقنيات الاخرى. اذا كنت تأمل فى methods افضل للتحكم فى الاخطاء من عبارات if المتداخلة، و الـ Boolean flags، وقيم ادخال الـ function الخاصة، فإن التعامل مع الـ exception هو طريقك .

أين تنشأ الـexceptions ؟:

يمكن ان تأتى الـExceptions من مصادر مختلفة على سبيل المثال، يستطيع برنامجك ان يولد exception للظروف الغير عادية . تستطيع Delphi componentsان تولد exceptions للعديد من الـ events مثل تعيين قيمة خارج النطاق لخاصية ما، أو محاولة فهرسة عنصر array غير موجود .

ان الـ functions و procedures الـ Run-time library يمكن ايضاً ان تنتج exceptions وان تنفيذ تعبير رياضى عملية غير مسموح بها مثل القسمة على صفر يضمن ظهور exception والـ methods الاخرى للعمليات المنتجة للـ exception تشمل الاشارة لـ nil pointer، تخصيص المزيد من الذاكرة أكبر من أكبر block خالى متاح، وتنفيذ تعبير type-cast غير مسموح به .

اذن، كل عبارة فى برنامج قد تؤدى الى exception. ولكن، بعض العبارات تعتبر آمنه نسبياً، وجزء كبير من مهممتك فى كتابة code قوية هو تحديد أى العبارات تحتاج الى حماية وأيهما لا يسبب ضرراً. ونماذج هذا الباب تكشف الحالات النموذجية التى يمكن ان تساعدك على اتخاذ هذه القرارات لبرامجك .

ولكن، هذا لا يعنى انه يتعين عليك ان تكتب عبارات للتعامل مع كل exception ممكن . ان معالجة الـ exception العامة لـ Delphi، والتى تم اضافتها لكل تطبيق، تتولى بصورة تلقائية معالجة الـ exception بعرض الـ dialog box مع وصف للمشكلة وموضع العبارة الخاطئة وهذه الاستجابة الافتراضية حسب النظام

تعتبر كافية للبرامج الاختبارية والامثلة كاغلب تلك الموجودة فى هذا الكتاب ، ولكنك تحتاج ان تكتب exception handlers للتعامل مع متطلبات برنامجك الفريدة .

بعض انواع العمليات التى يمكن ان ينتج عنها exception تشمل مايلى :

- التعامل مع ملف .
- تخصيص أماكن للذاكرة .
- Windows resources .
- الـ Objects و الـ forms التى تنشأها فى وقت التشغيل .
- التداخل بين نظام التشغيل ومكونات الأجهزة .

:Exceptional keywords

يوفر الـ Object Pascal العديد من الكلمات الاساسية الخاصة بإنشاء والتعامل مع الـ exceptions. وهذه الكلمات الاساسية هىtry ، except ، on-do-else، finally ، raise ، وat . إنك ستقابلهم جميعاً فى هذا الباب . لاتحاول استخدام هذه الكلمات لاغراض اخرى .

ملحوظة:ان الـ Turbo Pascal compilers والـ Borland الأوائل لا يدركوا كلمات الـ exception الاساسية للـ Object Pascal . ان التعامل مع Exception تقنية جديدة فى الـ Object Pascal compiler الخاص بـ Delphi .

بعض المصطلحات الجيدة:

فى الفقرات السابقة ، استخدمت بعض المصطلحات الجديدة مثل exception و handler دون توضح معانيها فيما يلى بعض التعريفات لهذه المصطلحات وغيرها من العبارات التى تحتاج الى معرفتها قبل استخدام الـ exception فى تطبيقك :

- **Exception:** هو ظرف غير عادى أو غير متوقع يعترض المسار الطبيعى للبرنامج .

- **Raising an exception**: هو الاشـارة الى ظرف غيـر عـادى أو غيـر مـتـــوقع . ان الـ Component methods، والـ run-time library، اخطاء الاجهزة–حتى تعيين قيمة غير مسوح به لخاصية Component– يمكن ان تُحدث exception. ويستطيع تطبيقك ايضاً ان يحدث exception عند اكتشاف سبب يتطلب تعاملاً خاصاً . استخدم الكلمة الاساسية raise لتحدث exception .

- **Exception handler**: هو code يـحل المشـكلة الـذى أحـــــدثـت exception . وهذا الـ code يجب ان يعيد النظام الى حالة مستقرة حتى يستطيع البرنامج ان يستمر فى التشغيل بشكل طبيعى .

- **Exception instance**: هو object من class تنحـــدر غــالبــاً من الـ Exception classومعرفة فى الـ .SysUtils unit . وتحتوى حالة الـ exception على معلومات تصف طبيعة الظرف الاستثنائى . ان حدوث exception يخصص ذاكرة لحالة الـ exception . و تحرر الـ handle exceptions بصورة تلقائية الذاكرة المخصصة .

- **Try block**: هو عبارة واحدة أو أكثر ، تسبقها الكلمة الاساسية try، تريد ان تعالج exception فيها أو تريد ان تحرر resource مثل تخصيص الذاكرة منها .

- **Except block**: هو عبارة واحـدة أو أكثرتعالج exception الذى حدث بواسطة عـبـارات فى الـ try block السـابق . تبـدأ الـ Except blocks بالكلمـة الاساسية except، وقد تحتوى اختيارياً على عبارات on-do-else . يجب ان يتبع الـ except block الـ try block مباشرة .

- **Finally block**: هو عبارة واحدة أو أكثر يجب ان تنفذ لتحرير الذاكرة المخصصة ، لاغلاق ملفات ، أو لأداء مهام ضرورية اخرى فى الـ event الخاص بالـ exception الذى حـدث بواسطة عـبـارات فى الـ try block السـابق . تبــدأ الـexception raised بعـبـارات فى الـ try block السـابق . تبــدأ الـ Finally blocks بالكلمـة الاساسية finally ويجب ان يتبعوا الـ try block مباشرة . لا تعالج الـ Finally blocks exceptions فقط الـ except blocks يمكن ان تفعل هذا .



- **Protected-statement block**: هي عبارة واحدة أو أكثر لها handlers خاص لأي exceptions يمكن ان ترفعها تلك العبارات . ان الـ protected-statement block يتكون من زوج من الـ try block والـ except .

- **Protected-resource block**: هي عبارة واحدة أو أكثر لها deallocation handlers معينة لأي exceptions يمكن ان تحدثها استخدام الـresources . وامثلة resources تشمل تخصيص أماكن للذاكرة ، الـ Windows resources ، والملفات . يتكون الـ protected-resource block من زوج من الـ try block والـ finally .

- **Robust application**: هو برنامج يستخدم blocks عبارات محمية و resource محمية لمعالجة كل ظروف الخطأ الممكنة التي قد تحدث ، وحتى يزيل تخصيص الـ resource بسلام .

Note

ملحوظة: تذكر هذه القاعدة : ان robust application يأخذ وقتاً فقط وهو لا يترك آثاراً . والهدف من استخدام exceptions هو كتابة code قوى تتولى بعناية كل ظروف الخطأ الممكنة .

الـProtected-statement blocks:

أداتك الاساسية لمعالجة الـ exceptions . توضح القائمة (١٩-١) التخطيط لإنشاء protected-statement block باستخدام الكلمات الاساسية try و except . لاحظ ان الـ block ينتهي بالكلمة الاساسية end وفصلة منقوطة . وتشير علامات التعليق تشير الى نوع العبارات التي يمكنك ادخالها في المواضع المختلفة .

(القائمة للمعلومات فقط وليست على القرص المدمج) .

القائمة (١٩-١): تخطيط لإنشاء protected-statement block

```
{ Unprotected statements }
try
  { Protected statements that may raise an exception}
except
  { Statements to handle any raised exceptions }
end;
   { More unprotected statements }
```

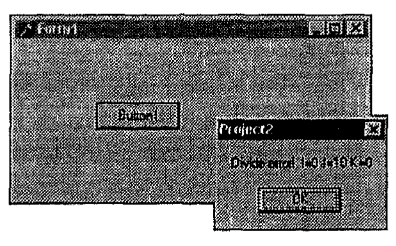
جرب مثالاً حقيقياً لتعرف كيفية استخدام التخطيط الموجود فى القائمة (١٩-١) وقم بإنشاء protected-statement block . اتبع الخطوات التالية :

١- ابدأ تطبيقاً جديداً .

٢- أضف Button object على الـ form .

٣- اضغط مرتين Button1 ، واستخدم القائمة (١٩-٢) كارشاد فى كتابة برنامج OnClick مع protected-statement block . ان نص هذه القائمة موجود فى ملف Except1.pas على القرص المدمج فى دليل الـ Source\ExceptMisc .

٤- قم بـ Compile وتشغيل البرنامج بضغط F9 . اضغط زر البرنامج لتجبر الـ exception على الحدوث . يوضح شكل (١٩-١) dialog الرسالة المعروضة بواسطة عبارة الـ exception handler ShowMessage التى تراها عندما تضغط الزر .



شكل (١٩-١): الـ OnClick للـ Except1.pas يعرض هذه الرسالة عندما يتعامل exception مع القسمة على صفر

القائمة (١٩-٢): ExceptMisc\Except1.pas ان OnClick Button هذا يعرض كيفية انشاء protected-statement block باستخدام الـ try والـ except

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I, J, K: Integer;
begin
  I := 0;
    J := 10;
    try
   K := J div I;
```

```
      except
        ShowMessage(
            'Divide error! '+
          ' I=' + IntToStr(I) +
          ' J=' + IntToStr(J) +
          ' K=' + IntToStr(K) );
        end;
    end;
```

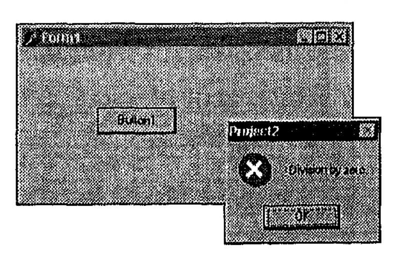
Tip **فكرة:** عند اختيار الـ exceptions، من الافضل ان تغلق WinSight32 message-spy utility التابعة لـ Delphi اذا كانت تعمل. اذا كنت تستخدم Delphi 4، اختر خيارات الـ Tools|Debugger، اختر صفحات exceptions اللغة، وألغ اختيار الخيار "Stop on Delphi Exceptions" . اذا كنت تستخدم Delphi 3، اختر الـ Preferences page tab، وابطل الخيار Break on Exception . كذلك انظر فصل "اعتبارات البيئة" في هذا الباب لمعرفة المزيد عن استخدام هذه الخيارات لازالة أخطاء exception handling البرنامج. ولكن اثناء تعلم كيفية exception، من الافضل ان تبطل وسائل تقرير الـ exception الزائدة هذه. بهذه الطريقة، إنك ترى بدقة كيف يستجيب البرنامج لنظام المستخدم النهائي. (بالتبادل، يمكنك compile البرنامج وتشغيل ملف الـ exe الخاص به باستخدام الـ Windows Explorer).

ان الـ procedure الموجودة في القائمة (١٩-٢) ينفذ معادلة رياضية لأكل البازلاء بالسكين. ان التعبير J div I يحاول ان يقسم عشرة على صفر، مما يجعل الـ run-time library للـ Object Pascal تحدث exception. ولكن لان التعبير موجود في try block، الـ procedure يعالج الـ exception بعرض رسالة. ان عبارة الـ ShowMessage في الـ except block تنفذ فقط اذا كانت عبارة واحدة أو أكثر في الـ try block السابق فاشلة. يمارس البرنامج تحكما تماماً في أي exception- اذا دخل البرنامج الـ except block ، تكون قيمة الـ K غير معرفة بالطبع، هذا مجرد مثال بسيط، ولكن في الـ code المتداخلة بعمق والمعقدة تكون مثل هذه المعلومات لاقيمة لها.

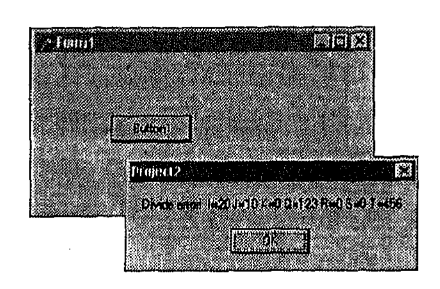
لعرض ما يحدث عندما تحدث الـ unprotected-statement exception، احذف الـ try، الـ except والـend الأولى من القـائمة (١٩-٢). اترك العبارات الاخرى كما هى، ثم اضغط F9 لـ compile وتشغيل البرنامج المعدل. اضغط الزر لترى dialog الذى يعرضه برنامج الـ exception handler الافتـراضى حسب نظام Delphi، والموضح فى شكل (١٩-٢). هذه التجربة توضح انه حتى اذا لم تستخدم protected-statement blocks، يتم معالجة كل الـ exception فى النهاية. ان الاستجابة الافتراضية حسب النظام لتلك الـ exception قد لا تكون كافية لتطبيق، ولكن على الاقل ان البرنامج ليس على اسوأ حال بلا قيمة. ان الاستجابة الافتراضية تكون دائماً افضل من لا شئ.

On the CD-ROM

على القرص المدمج: عندما يحدث exception فى الـ try block، يقفز التنفيذ على الفور الى العبارة الاولى فى الـ try block. لا يتم تنفيذ العبارات الاخرى فى الـ except block بعد تلك التى exception، وهذه الحقيقة قد يكون لها عواقب هامة. على سبيل المثال، فكر فى protected block فى القائمة (١٩-٣)، والذى يمكنك ادخاله فى للـ OnClick Button كما فى الاختبار السابق. (يوجد الـprocedure على القرص المدمج فى دليل Source\ExceptMisc فى ملف Except2.pas). اذا فشل أى من تعبيرات القسمة لأى سبب، يقفز التنفيذ على الفور للـShowMessage، بعدها يستمر الـprocedure بشكل طبيعى بعد نهاية الـ protected block . على سبيل المثال، اذا فشلت القسمة الثانية، وهذا ماسيحدث مع القيم المعينة المتغيرات الـprocedure، لا تنفيذ العبارة الثالثة فى الـ try block. يوضح شكل (١٩-٣) dialog الرسالة المعروضة بواسطة عبارة الـ ShowMessage للـ procedure .



شكل (١٩-٢): برنامج الـ exception handler الافتراضى لتطبيق Delphi يعرض displays لأى unhandled exceptions



شكل (١٩-٣): برنامج Except2.pas OnClick يعرض هذه الرسالة عندما يحدث exception

القائمة (١٩-٣): ExceptMisc\Except2.pas. اذا أحدثت أي واحدة من عبارات القسمة الثلاثة exceptions، يقفز التنفيذ مباشرة الى الـ ShowMessage

```
procedure TForm1.Button1Click(Sender: TObject);
var
    I, J, K, Q, R, S, T: Integer;
begin
 I := 20; J := 10; Q := 123; T := 456;
    try
     K := (I div J) - 2;
       R := Q div K;  // ???
       S := T div R;
      except
     ShowMessage(
      'Divide error! '+
        ' I=' + IntToStr(I) +
        ' J=' + IntToStr(J) +
        ' K=' + IntToStr(K) +
        ' Q=' + IntToStr(Q) +
        ' R=' + IntToStr(R) +
        ' S=' + IntToStr(S) +
        ' T=' + IntToStr(T) );
      end;
end;
```

على القرص المدمج: تعرض القائمة (١٩-٣) واحدة من المميزات الرئيسية لمعالجة الـ exception-لازاله الخطأ المتكرر بفحص العبارات الفورية المرتبطة السابقة أو التالية . من المفيد ان نقارن التشابه بين الـ error-handling و الـ non-exception، والذى يوجد له objects لا تحصى . توضح القائمة (١٩-٤) حالة فوضوية نموذجية . لا تكتب ابداً code مثل هذا . يوجد الـ procedure على القرص المدمج فى ملف الـ Except3.pas فى دليل Source\ExceptMisc . يمكنك اختباره باستخدام procedure مثل الـ Button OnClick يعرض الـprocedure displays الرسالة نفسه الموضح فى شكل (١٩-٣).

القائمة (١٩-٤): ExceptMisc\Except3.pas . مع exception handling إنك لا تحتاج الى كتابة code فوضوية مثل هذه

```
procedure TForm1.Button1Click(Sender: TObject);
var
    I, J, K, Q, R, S, T: Integer;
    { Sub-procedure for displaying error message }
    procedure ReportError;
    begin
      ShowMessage(
          'Divide error! '+
           ' I=' + IntToStr(I) +
           ' J=' + IntToStr(J) +
           ' K=' + IntToStr(K) +
           ' Q=' + IntToStr(Q) +
           ' R=' + IntToStr(R) +
           ' S=' + IntToStr(S) +
           ' T=' + IntToStr(T) );
        end;
begin
    I := 20; J := 10; Q := 123; T := 456;
    if J = 0 then ReportError else
    begin
        K := (I div J) - 2;
```

```
        if K = 0 then ReportError else
          begin
            R := Q div K;
            if R = 0 then ReportError else
            begin
              S := T div R;
             end;
        end;
      end;
    end;
```

اسئلة نموذجية:

فيما يلي ثلاث نقاط يجبن عن بعض الاسئلة النموذجية والتي قد تكون لديك على كيف تؤثر الـ protected-statement blocks على سير البرنامج :

- اذا كانت عبارة في الـ except block تعالج exception يستمر الـ procedure أو الـ function بشكل طبيعي بعد نهاية الـ protected block .
- اذا لم تتولى أي عبارة exception handles يخرج الـ procedure أو الـ function الحالية فوراً، ويمر الـ exception من خلال سلسلة الاستدعاء الى ان يجد الـ handler المناسبة .
- ان الـ Unhandled exceptions تصل في النهاية الى الـ handles exception الافتراضي للتطبيق، والذي يعرض dialog ورسالة . ان handles exception الافتراضي هو الـ handles على أي exception لم توفر لها handles . لا يجب عليك ان تكتب exception handles الافتراضي حسب النظام- ان Delphi يلحقه تلقائياً بكل تطبيق عمل compile له . مؤخراً في هذا الباب، سوف اشرح كيفية استبدال الـ handles الافتراضي بـ code خاصة بك، والذي لا يكون ضرورياً، ولكن قد يكون نافعاً في المشروعات المتقدمة .

: Protected-resource blocks

ان عرض رسائل الخطأ يعد جانباً واحداً في الـ exception handling ويجب ايضاً على الـ robust application ان يستعيد الاستقرار عندما تحدث

الكارثة . على سبيل المثال، اذا حدث خطأ قرص، يجب ان يحرر التطبيق أى ذاكرة مخصصة والتى قد تشغل مساحة بلا داعى حتى يعيد المستخدم الاقلاع . يتخلص البرنامج القوى من الاخطاء باغلاق الملفات المفتوحة deallocating Windows resources، وفعل كل ما هو ممكن لاستعادة النظام فى وسط الفوضى .

استخدم الكلمات الاساسية try و finally لإنشاء protected-resource blocks . توضح القائمة (١٩-٥) التخطيط الاساسى، والذى يشبه الى حد كبير protected-statement blocks . فى الواقع، ان الاختلاف الوحيد هو فى كلمة finally مكان كلمة except . ولكن، بالرغم من التشابه الظاهرى، تختلف protected-resource blocks بشكل كبير فى العمل عن protected-statement blocks . (نص هذه القائمة لغرض العرض فقط وليس موجود على القرص المدمج) .

القائمة (١٩-٥): تخطيط لإنشاء protected-resource blocks

```
{ Allocate memory or other resource}
try
    { Statements that may raise an exception }
finally
  { Free the resource-guaranteed to execute!}
end;
{ Continue if no exceptions occurred in the try block }
```

ان العبارات الموجودة فى الـ finally block تنفذ دائماً بغض النظر عما اذا كانت ايه عبارات فى الـ try block تحدث exception . من الناحية النموذجية، أن العبارات الموجودة فى الـ finally block تحرر الذاكرة ويغلق الملفات، تؤدى عمليات اخرى يجب القيام بها والتى تستعيد الاستقرار الى النظام عندما يقع exception .

Caution

تحذير: ان العبارات الواقعة خارج الـ try block والتى تستطيع ان تحدث exception تجعل الـ procedure أو الـ function تخرج على الفور، متخطية الـ finally block . لضمان تنفيذ الـ finally block، ضع فى الـ try block كل العبارات التى قد يحدث exception وتترك الـ resource المخصص .

ان التـخـبـط العـام فى الـ protected-resource blocks هى اين توضع العبارة التى تخصص resource. فبالرغم من ان هذا قد يحدث resource (على سبيل المثال، اذا فشل تخصيص الذاكرة نتيجة نقص الـ RAM)، فإن عبارة التخصيص لا ينتمى الى الـ try block. وهذه القاعدة غاية فى الاهمية. وإحدى الطرق لتذكر القاعدة هى ان تحفظ دائماً ان الغرض من الـ finally block هو إزالة تخصيص الـ resource ، وانت لا تريد ان تزيل تخصيص شئ لم يكن مخصصاً فى المقام الاول. لذلك، يجب ان تضع عبارة التخصيص قبل الـ try block. وداخل ذلك الـ resource، إدخل أى عبارات قد تحدث exception، مما يجعل الـ procedure أو الـ function ان تخرج تاركة الـ resource المخصص متأرجحاً فى مكان ما فى الذاكرة حتى يعيد المستخدم الاقلاع.

ويوضح مثالاً بسيطاً هذه التقنية الهامة، ويظهر كيفية استخدام protected-resource blocks للحماية دون وجود resources متارجحة، والتى قد تقضى على RAM فى نظامك. ان اختفاء الايقونات، والنص الذى يتحول بطريقة غريبة الى الـ System font، ولوحة المفاتيح الغير مستجيبة، والفوضى العامة هى اعراض تسرب resources ، يحدث عادة بواسطة تأرجح الذاكرة وتخصيصات الـ resources الاخرى.

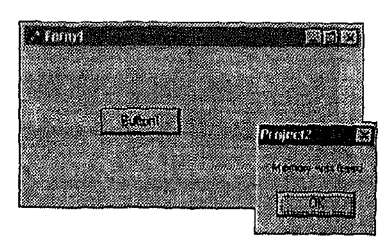
جرب هذه الخطوات :

١ - ١ - ابدأ تطبيقاً جديداً.

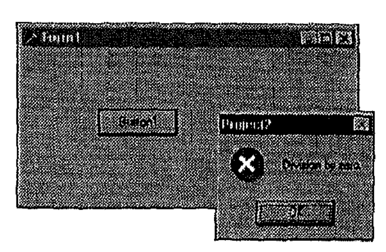
٢- أضف Button على الـ form.

٣- اضغط الـButton مرتين لإنشاء OnClick ، وإملأ procedure باستخدام القائمة (١٩-٦) كمرشد لك. (هذا النص موجود على القرص المدمج فى ملف Except1.pas فى دليل الـ Source\ExceptMisc).

٤- اضغط F9 لـ Compile وتشغيل البرنامج ثم اضغط زر لاختبار ما يحدث فإذا حدث exception فى procedure يخصص قالباً من الذاكرة. شكل (١٩-٤) وشكل (١٩-٥) يظهراً dialog الرسالة التى تراها بالترتيب عند تشغيل الـ procedure الاختبارى.



شكل (١٩-٤): بالرغم من ان الـ exception يحدث فى الـprocedure الاختبارى Except1.pas ، الا ان dialog الرسالة هذا وذلك الموضح فى شكل (١٩-٥) يثبتا ان الذاكرة المخصصة قد تم تحريرها بشكل سليم



شكل (١٩-٥): يعرض الـ Except1.pas هذا الـ dialog ا بعد ان تغلق ذلك الموضح فى شكل (١٩-٤)

القائمة (١٩-٦): ExceptMisc\Except4.pas . OnClick Button هذا يوضح كيفية إنشاء protected-resource block باستخدام الـ try والـfinally.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I, J, K: Integer;
     P: Pointer;
begin
  I := 0;
     J := 10;
     GetMem(P, 4098);  // Allocate memory resource
     try
       K := J div I;  // Raises an exception!
          ShowMessage('Results: ' +
      ' I=' + IntToStr(I) +
            ' J=' + IntToStr(J) +
            ' K=' + IntToStr(K) );
```

```
    finally
      FreeMem(P, 4098);  // Guaranteed to execute
        ShowMessage('Memory was freed');
      end;
  end;
```

تشبه القائمة (١٩-٦) الـprotected-statement block، ولكن تضيف عبارة تخصيص ذاكرة تستدعى الـ GetMem للاحتفاظ بـ ٤٠٩٨ بايت من الـ RAM لاحظ ان هذه العبارة (وكذلك تعينات متغير العدد الصحيح الغير ضارة) قد تم وضعها خارج الـ try block . وداخل الـ try block يوجد تعبير القسمة الخاطئة . اذا لم تكن هذه القسمة داخل الـ try block، واذا كان تنفيذ تلك العبارة تحدث exception، فقد ينتهى البرنامج على الفور، تاركا block الذاكرة المخصصة ضائعاً فى الفضاء .

ان الـ finally block يحمى من هذا الخطأ باستدعاء الـ FreeMem بغض النظر عما اذا كانت القسمة تحدث exception ام لا . (قم بتعيين اثنين للـ I واعد تشغيل الاختبار لتثبت ذلك) . ان عبارة الـ ShowMessage الاخيرة هى لمجرد الايضاح-عملياً، تقوم الـfinally blocks غالباً بخدماتها فى هدوء .

Note **ملحوظة:** تذكر : ان العبارات فى الـ finally block مضمون تنفيذها حتى اذا تم رفع أى exception فى الـ try block السابق .

ان الـfinally blocks لا تعالج الـ exception فقط ولكن الـ except block هى تفعل هذا . الـfinally blocks يضمن تنفيذ عبارات بغض النظر عما اذا كانت عبارات اخرى موجودة فى الـtry blocks السابق تحدث exception ام لا . فى هذه الحالة، لا يتم الـexception handler، وبذلك، عندما تقوم بتشغيل الـ code فى القائمة (١٩-٦)، يعرض برنامج الـexception handler الافتراضى لـDelphi dialog رسالة خطأ . ولكن، كما ستوضح الفصول التالية، يمكن للـprocedure ان يكون لديه protected-statement و protected-resource blocks.

الـtry-except والـtry-finally blocks المتداخلة:

اذا كنت مازالت فى بداياتك مع الـexception handler ، فقد تتساءل كيف يمكنك الـexception handler وحماية تخصيصات الـprocedure فى نفس الـprocedure . والحلية هى ان تتذكر ان هذين يعتبران عمليتان منفصلتان تماماً ، وانك لا يمكن ان تدمج الـtry-except والـtry-finally blocks . فـلا يوجـد object مـثل الـtry-except-finally block فى الـObject Pascal الخـاص بـDelphi .

ولكن ، يمكنك ان تدخل الـtry-except block فى الـtry-finally blocks لتعالج الـexceptions وتمنع تأرجح الـresources . توضح القائمة (١٩-٧) التخطيط الاساسى لانجاز هذا المستوى الاضافى من الحماية . (هذا . الـcode للعرض فقط وغير موجودة على القرص المدمج) .

القائمة (١٩-٧): تخطيط لإنشاء protected-statement و protected-resource blocks متداخلة

```
{ Allocate resource }
try
  try
      { Statements that might raise exceptions }
    except
   { Statements that handle exceptions }
    end;
finally
  { Free the resource }
end;
```

إنك تحتاج الى استخدام الـtry-except block والـtry-finally المتداخلة كما هو موضح فى القائمة (١٩-٧) فقط عندما تعالج الـexcept block نوع معين من الـexception ، وفى هذه الحالة ، فان الانواع الاخرى من الـexception تجعل الـprocedure ينتهى مبكراً . فى هذا المثال والامثلة السابقة ، يعالج الـexcept block كل الـexception التى تحدث فى الـtry block السـابق . ولكن كـمـا

سـأوضح مـؤخـراً فى هذا البـاب ، إنك غـالبـا مـا تقـبض على بعض الـ exception وتتـرك الاخـرى تمر الى أعلى فى سلسلة الاسـتـدعـاء ، وربما تصـل الى برامـج الـ exception المعالجة الافتراضية لـ Delphi فى غياب الـtry-finally block ، فان تلك الـ exception الاخرى ، لانها لـم يتم علاجـهـا فى الـ procedure ، قد تجعل تخصيصات الذاكرة لا يتم تحريرها .

على القرص المدمج: موخراً فى هذا الباب ، سوف اشرح كيفية معالجة انواع معينة من الـ exception باستخدام عبارات الـ on-do . اما الآن ، فـإن القـائمـة (١٩-٨) توضـح مـثـال قابل للتنفـيـذ للتـخطيط الوارد فى القـائمـة (١٩-٧) . توضـح القـائمـة (١٩-٩) نفس الـcode مع الـ try-except block المتداخل وقـد تم ابعاده الى function قابلة للاستدعاء ، والتى تعتبر اقل فاعلية ، ولكن تلغى التداخل الذى يجده بعض المطورين امراً مجيراً . جرب كلا الـ two procedures كـمـا فعلت فـيـمـا سـبق باضـافـةButtons على form وانشـاء OnClick . اضـغـط F9 للـ compile والتـشـغـيل ، ثـم اضـغـط الزر . تثبت رسـائل البـرنامج ان الذاكـرة المخـصـصـة قـد تم تحـريرها بالرغم من ان هناك exception يحدث . توجد القائمتان فى ملفى Except5.pas و Except6.pas وعلى القرص المدمج فى دليل Source\ExceptMisc .

ان Button OnClick فى القـائمـة (١٩-٨) يظهـر كـيـفـيـة تداخـل الـ try-except block فى الـ try-finally block لمعـالجـة exceptions ولمنع الـ resources من التأرجح .

القائمة (١٩-٨): ExceptMisc\Except5.pas

```
procedure TForm1.Button1Click(Sender: TObject);
var
 I, J, K: Integer;
    P: Pointer;
begin
 I := 0;
    J := 10;
    GetMem(P, 4098);
    try
```

```
try
    K := J div I;
      ShowMessage('Results: K=' + IntToStr(K));
     except
    ShowMessage('Divide error! ' +
      ' I=' + IntToStr(I) +
      ' J=' + IntToStr(J) );
end;
  finally
FreeMem(P, 4098);
     ShowMessage('Memory was freed');
end;
end;
```

ان OnClick و function فى القائمة (١٩-٩) يعتبران من الناحية العملية متطابقان للـ code الموجودة فى القائمة (١٩-٨)، ولكنها تصنع الـtry-except المتداخل داخل function قابلة للاستدعاء، وهى الـ GetInt. والنتيجة هى OnClick procedure اوضح على حساب استدعاء function اضافى.

القائمة (١٩-٩):ExceptMisc\Except6.pas

```
function GetInt: Integer;
var
 I, J, K: Integer;
begin
 I := 0;
    J := 10;
    try
     K := J div I;  // Raises an exception
        Result := K;   // Assign function result (doesn't
     execute)
        ShowMessage('Results: K=' + IntToStr(K));
      except
     Result := 0;   // Assign function result on error
```

```
        ShowMessage('Divide error! ' +
          ' I=' + IntToStr(I) +
            ' J=' + IntToStr(J) );
          end;
end;

procedure TForm1.Button1Click(Sender: TObject);
var
 K: Integer;
    P: Pointer;
begin
 GetMem(P, 4098);  // Allocate memory resource
    try
      K := GetInt;    // Might cause an exception
        finally
   FreeMem(P, 4098);  // Guaranteed to execute
        ShowMessage('Memory was freed');
        end;
end;
```

توضح القائمة (١٩-٩) كيف تقوم معالجة الـ exception بفك ازدواج عبارات برنامج عادية من منطق معالجة الخطأ الخاص بها. على سبيل المثال، احذف try، و except ، وكل العبارات فى قالب الـ except و الـand الأولى من الـ GetInt function . تبدو الـfunction مثل هذه:

```
begin
 I := 0;
    J := 10;
    K := J div I;  // Raises an exception and exits
   procedure!
    Result := K;
    ShowMessage('Results: K=' + IntToStr(K));
end;
```



اضغط F9 ولاعادة تشغيل البرنامج، ثـم اضغط الزر لتخبر الـ exception على الحدوث . ان الـ Button OnClick مازال يحرر الذاكرة المخصصة بالرغم من ان الـ function لم تعد تعالج الـ exception الذى حدث بنسب القسمة الخاطئة . اذا كان OnClick لم يستخدم الـ try-finally، فإن الـ GetInt's exception قد يجعل الـ OnClick procedures ينتهى، تاركاً الذاكرة المخصصة تتأرجح .

معالجة الـException:

قد مت الفصول السابقة اغلب تقنيات معالجة الـ exception اللازمة لكتابة اللازمة لكتابة تطبيقات Delphi قوية . فى هذا الجزء، سوف تتخطى الاساسيات لتعرف كيفية معالجة انواع خاصة من الـ exception، وكيفية حدوثه وحدوث exception خاصة بك . ولكن اولاً، انك بحاجة الى التفكير فى بعض الخيارات التى تؤثر فى كيفية استجابة نظام التطوير الخاص بك للـ exception .

عند تطوير تطبيقات- وخاصة debugging exception handlers - فمن الأفضل أن تجعل Delphi يوقف البرنامج عندما يحدث exception . لتشغيل هذا الميزة، اختر امر الـ ...Tools| Environment Options . اذا كنت تستخدم Delphi 3، اختر الـ Preferences page tab، وقم بتشغيل الـ Break on Exception check box . اذا كنت تستخدم Delphi 4، اختر خيارات الـ Tools|Debugger، اختر صفحة exception اللغة، وابطل اختيار الـ Stop on Delphi Exceptions check box .

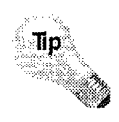
ملحوظة: بعض الاوامر المشار إليها هنا تختلف اعتماداً على نسختك من الـ Delphi . اذا كنت تستخدم Delphi Version 1، اختر الـ Environment|Options؛ وللـ Version 2 ، اختر Tools|Options، ثم اتبع التعليمات السابقة للـ Version 3. من Delphi .

حاول تشغيل برنامج اختبار exception . (واحداً يحتوى على الـ event handler للقائمة (١٩-٢)، مثلاً) . مع اختيار وابطال هذه الخيارات . عند تشغيلها، يعرض الخيار تقريراً كاملاً عن أى exceptions، ويتوقف البرنامج عند العبارة التى سببت المشكلة . يوضح شكل (١٩-٦) الـ dialog box الناتج الذى يعرضه Delphi . والسطر الذى سبب المشكلة يتم ابرازه فى نافذة محرر الـDelphi

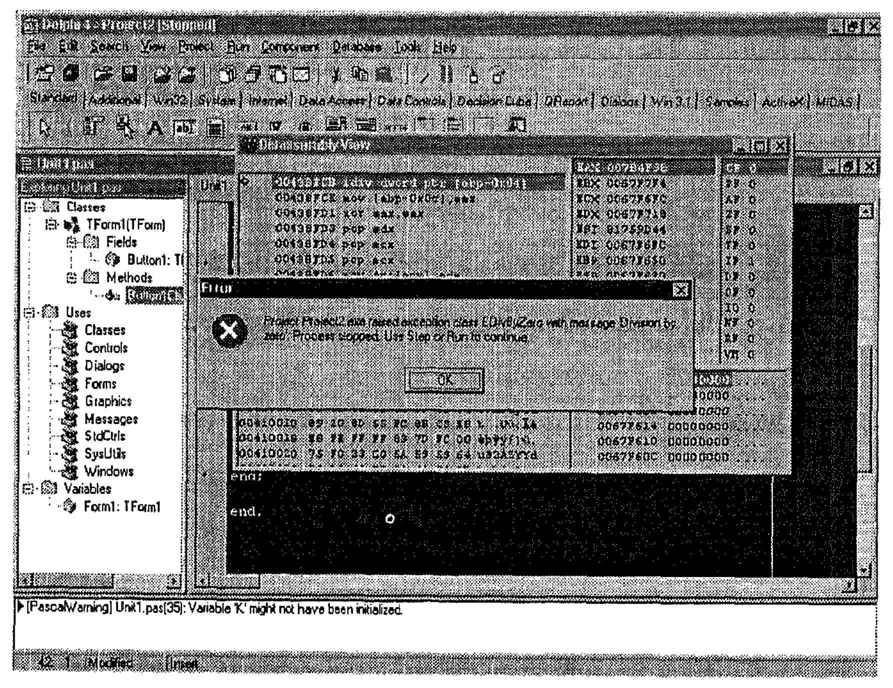
code . من المهم ان تفهم ان Delphi نفسه ، وليس برنامج معالجة الـ exception الافتراضى للتطبيق ، يعرض نافذة الـ dialog هذه .

يجب ان تقوم بتشغيل برنامجك داخل Delphi (بمعنى اوضح ، فى نمط الـ dialog) لتراه . ان المستخدمين النهائيين لتطبيقك لايرون ابداً هذا الـ dialog .

اقرأ معلومات الـ dialog ، واضغط زر الـ OK . اختبر العبارة التى تم ابرازها والتى أحدثت exception . مازال البرنامج يعمل ، ولكنه يتوقف عند العبارة المتهمة . اضغط F9 لاستمرار التشغيل ، أو يمكنك ان تضغط F8 أو F7 لتنقل خطوة واحدة بالبرنامج بدءاً من هذا الموضع .



فكرة: بالرغم من انك تنظر الى code برنامجك فى Delphi ، فمازال البرنامج يعمل فى نمط الـ dialog. يجب ان تقوم بتشغيل واخراج البرنامج قبل ان تستمر فى تطوير التطبيق .



شكل (١٩-٦): عندما يتم معالجة الـ exception بواسطة الـ debugger ويحدث exception يتوقف البرنامج ويعرض Delphi displays يشبه هذا

للاستمرار، اضغط F9 الآن. لقد انتقلت مرة اخرى الى نافذة البرنامج، واعتماداً على المثال الذى تقوم بتشغيله، فإنك ترى واحدة أو أكثر من يهشمخل الرسائل التى يتم عرضها بواسطة البرنامج نفسه. فى المرة الاولى التى تجرب فيها هذه الخطوات، فإن الـ dialogs قد تبدو محيرة، ولكن بعد ان تفهم المستويات المختلفة لمعالجة الـ exception التى تحدث، فإن المعلومات تصبح قيمة للغاية فى مطاردة الاخطاء- خاصة أى اخطاء فى برامجك المعالجة للـ exception.

عندما يتم معالجة الـ exception بواسطة الـ debugger، يمكنك تشغيل التطبيق باستخدام الـ Windows Explorer لاختبار كيف يستجيب البرنامج التام للاخطاء خارج Delphi. قم بتشغيل الـ code داخل Delphi للحصول على معلومات حول الـ exception بالإضافة الى أى معالجة الـ exception التى يوفرها تطبيقك.

Note

ملحوظة: لاستكمال هذا الباب، قد تود ان يكون لديك exception تعالج من خلال debugger، ولكنك تحتاج ان تضغط F9 مرة واحدة لتستمر فى تشغيل البرنامج فى كل مرة يحدث فيها الـ exception. اذا وجدت ان هذا متعب لك، اغلق معالجة الـ exception بالـ debugger. اذا كنت تستخدم Delphi 3، اختر الـ Preferences page tab، وقم بابطال الـ Break on Exception check box.

الـ exception instance:

كما ذكرنا، ان exception هو حدث أو ظرف يعترض السير الطبيعى للبرنامج. ولكن من الناحية التكوينية، الـ exception يعتبر object من class، مشتقه من الـ Exception class المعرفة فى الـ SysUtils unit. وهذا الـ object يسمى exception instance .

ان حدوث exception يؤدى الى انشاء exception instance . اذا كانت عبارة فى جزء الـ except من protected-statement block تعالج الـ exception، فإن البرنامج يدمر الـ exception instance بصورة تلقائية. اذا لم تتولى أى عبارة معالجة الـ exception، يقوم التطبيق بتمرير الـ exception instance الى أعلى فى سلسلة الاستدعاء حتى تجد معالجة مناسبة، أو حتى تصل الى برنامج exception handler الافتراضى حسب النظام.

ان الطريقة الوحيدة الآمنة لتدمير exception instance هى معالجة الـ exception. فمعالجة الـ exception تدمر الـ exception instance بصورة تلقائية. فى الفصول التالية، سوف تتعلم كيفية الاشارة الى exception instance. لا تقم ابداً بكتابة code لتحرر أو تدمر هذه الـ objects. ان محاولة تحرير exception instance يؤدى الى خطأ فادح بالتطبيق.

معالجة exception معينة:

ان exception instance تعرف نوع الـ exception الذى حدث. يوفر الـ Delphi عدة classes، كلها مشتقة من الـ Exception، والتى تصف طبيعة مشكلة معينة. يمكنك استخدام هذه المعلومة لتحسن استجابة برنامجك للـ exception، وقد تكتب برامج معالجة exception يتم اطلاقها بواسطة انواع معينة من المشكلات. لتنفيذ هذه المهام، استخدم عبارة الـ on-do داخل الـ except block.

تنفذ عبارة الـ on-do خدمتين: إنها تعرف نوع معين من الـ exception، ويمكن ان توفر reference الى الـ exception instance وتوفر الـ exception instance معلومات عن نوع المشكلة التى حدثت. يمكن لبرنامجك ان يستخدم هذه المعلومات فى حل المشكلة ولعرض ملاحظات مساعدة للمستخدمين.

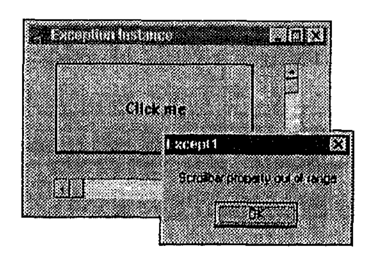
الإشارة الى حالة exception instance :

على القرص المدمج: ان التطبيق Except1 على القرص المدمج فى دليل Source\Except1 يوضح كيفية استخدام الـ on-do للاشارة الى exception instance يوضح شكل (١٩-٧) نافذة البرنامج الـ dialog المعروض عندما تضغط الزر. هذا يجبر الـ exception على الحدوث بمحاولة تعيين قيم غير مسموح بها للـ scrollbars controls التابعين للنافذة. توضح القائمة (١٩-١٠) OnClick للـ Button.

القائمة (١٩-١٠): Except1\Main.pas OnClick للـ Button

```
procedure TMainForm.Button1Click(Sender: TObject);
begin
  try
```

```
ScrollBar1.SetParams(0, 500, 0);
  ScrollBar2.SetParams(0, 500, 0);
 except
on E: Exception do
    ShowMessage(E.Message);
    end;
end;
```



شكل (١٩-٧) : ان ضغط زر الـ Except1 يجبر الـ exception على الحدوث ويعرض dialog الموضح هنا

يستدعى البرنامج الـ scrollbars SetParam ، ولكن لاجبار الـ exception على الحدوث ، فإنه يعكس متعمداً قيم الـMax والـ Min. يحدد الـ ScrollBar ان الـ Min يجب ان يكون اقل من أو مساو للـ Max ، والاستدعاء الاول للـ SetParams يحدث exception لهذا السبب.

فى الـ except block، تعرف عبارة on-do متغير E للـ Exception class . (يمكنك استخدام أى اسم، ولكن E اسم مناسب). وهذه العبارة لا تنشئ حالة Exception جديدة، إنها فقط تنشئ أشارة للـ (E) للـ exception instance التى تصف المشكلة. يستخدم البرنامج خاصية الـ Message string للـ E لعرض رسالة ذات معنى فى الـ dialog box.

ولكن، الايقاع بكل نوع exception بهذه الطريقة لا يكون دائماً افضل الحلول. والخطة الافضل من هذه هى الايقاع بـ exception معينة، وترك الاخرى تمر بطريقة عادية. بهذه الطريقة، تستجيب الـ code للـ exception التى تم اعدادها على ان تعالجها- parameter سيء، مثلاً، أو خطأ القسمة على صفر- ولكن تسمح برامج معالجة الـ exception الافتراضية لـ Delphi ان تعتنى بالمشكلات الاكثر خطراً مثل خطأ الخروج من الذاكرة.

الامساك بانواع معينة من الـ exception:

على القرص المدمج: يمكنك استخدام الـ on-do للايقاع بانواع معينة من الـ exception ، مثلاً، لمعالجة exception القسمة على صفر ، ولكن اسمح لبرامج معالجة اخرى أعلى فى سلسلة الاستدعاء ان تعالج انواع الـ exception الاخرى، كما هو موضح فى القائمة (١٩-١١). هذه هى نفس الـ code الموجودة فى القائمة (١٩-٢٠) ولكن هذه المرة، تعالج بشكل خاص فقط الـ exception EDivByZero . على القرص المدمج، يوجد نص الـ procedure . فى ملف Except7.pas فى دليل الـ Source\ExceptMisc .

القائمة (١٩-١١): الـ OnClick Butto يعرض كيفية انشاء rotected-statement block باستخدام الـ try والـ except لنوع معين من الـ exception

```
procedure TForm1.Button1Click(Sender: TObject);
var
 I, J, K: Integer;
begin
 I := 0;
    J := 10;
    try
      K := J div I;
        except
      on E: EDivByZero do
          begin
            ShowMessage(E.Message +
                ' I=' + IntToStr(I) +
                  ' J=' + IntToStr(J) +
                  ' K=' + IntToStr(K) );
               end;
  end;
end;
```

لاحظ ان عبارة الـ ShowMessage تعرض اولاً E.Message ، والتى تم تعريفها فى كل الـ exception objects على انها خاصية String. لا يجب عليك ان تعرض هذا الـ string، ولكنه متضمن فى كلexception objects، وفى اغلب الحالات، يجب ان يظهر dialog الخطأ فى مكان ما .

اختيارexception classes :

فيما يلى قائمة exception classes المعرفة فى الـ Delphi units المتنوعة. هذه القائمة ليست شاملة، ولكن تتضمن أغلب الـ exception التى غالباً ماستستجيب لها أو ترفعها فى تطبيقاتك. ان كل الـclasses فى الجدول مشتقة، وليس مباشرة بالضرورى، من الـ Exception class . استخدمها كما توضح القائمة (١٩-١١). على سبيل المثال، للامساك بخطأ EPrinter، استخدم code مثل هذه :

```
try
  { Printer operations }
except
  on E: EPrinter do
     ShowMessage(E.Message);  // Display printer error
     message
end;
```

- **EAbort**: exception يحدث بواسطة إستدعاء الـ Abort .

- **EAbstractError**: يتم حدوثه اذا حاولت عبارة ان تستدعى abstract method . يعرض Delphi تحذيراً اذا قام الـ code بانشاء instance من class ذات واحد أو أكثر من abstract methods . والمرة الوحيدة التى ستتلقى فيها الـ EAbstractError exception هى اذا تجاهلت هذا التحذير .

- **EAccessViolation**: يشير الى واحد من ثلاث أخطاء وصول غير مسموح الى الذاكرة: باستخدام الـ nil pointer ، أو بالكتابه الى صفحة code ، أو بالوصول الى صفحة ذاكرة غير مخصصة. لا يحدث هذا الـ memory exception ظاهرياً.

- **EArrayError**: يشير الى واحد من ثلاث عمليات array غير مسموح بها : فهرس خارج النظام، محاولة اضافة عنصر الى array ذا حجم ثابت، ادخال عنصر فى sorted array . انظر ايضاً EBitsError .

- **EAssertionFailed**: يحدث بواسطة الـ Assert عند تمرير تعبير False Boolean .

- **EBitsError**: يشير الى واحد ةمن اثنين من عمليات Boolean arrays الغير مسموح بها (باستخدام الـ TBits class) : قيمة فهرس سالبه أو خارج النطاق .

- **EClassNotFound**: هذا الـ exception يتم حدوثه عندما يحاول البرنامج قراءة object من class غير مرتبطة بالتطبيق (مثلاً، تم حذف تعريفها من تركيبه source code للـ form ، أو حاول البرنامج قراءة object غير معروف من مجموعة) .

- **EComponentError** : يحدث عند أى خطأ اثناء تسجيل الـ component يحدث ايضاً اذا حاول البرنامج ان يحصل على واجهة تطبيق الـ COM للـ component اذا لم يكن هذا الـ component يؤيد الـ COM . انظر الباب العشرين لمزيد من المعلومات عن انشاء وتشغيل الـ component .

- **EControlC**: يخص التطبيقات فقط . هذا الـ exception يتم حدوثه اذا قام مستخدم البرنامج بضغط Ctrl+C . هذا الـ exception لا يحدث ابداً فى تطبيق Windows .

- **EConvertError**: يحدث لاخطاء التحويل مثل محاولة تحويل string ذا رمز غير رقمى الى عدد صحيح باستدعاء الـ StrToInt .

- **EDatabaseError**: تحدث exception نتيجة لخطأ قاعدة البيانات العام بواسطة الـ Data Access والـ Data Controls . لمعرفة المزيد عن بعض exception قاعدة البيانات ، ابحث فى online help التابعة لـ Delphi عن الـ EDB exception classes ، مثلاً، EDBClient و EDBEditError .

- **DateTimeError**: يحـدث بواسطة اوقـات وتواريخ غـيـر صـحـيـحـة يتم ادخالها فى الـ TDateTimePicker object.
- **EDivByZero**: عدد صحيح يحاول القسمة على صفر .
- **EFCreateError**: يشير الى خطأ فى انشاء ملف .
- **EFilerError**: يحــدث بســبب المشــاكل التى تحــدث اثناء عــمليــات الـ file-stream.
- **EInOutError**: خطأ فى إدخـــال/ إخـــراج ملـــف . ان حـــقـــل الـ exception instance ErrorCode Integer يحــمل code خطأ I/O . وهذا الـ exception يتم حدوثه فقط فى الـ code compiled مع مفتاح الـ {$I+} فى حالة التشغيل (هذا هو التحديد الافتراضى). والـ codes الخاطئة هى : ٢- لملف غير موجود، ٣- لاسم ملف غير صحيح، ٤- للفات عديدة مفتوحة، ٥- access denied، ١٠٠- نهاية ملف، ١٠١- قرص ممتلئ، و ١٠٦- أدخال بيانات غير صـحـيـح . و الـ code الخـاطئـة الاخـرى ممكنة- انظر كـتـيب الـ MS-DOS Technical Reference للـ Microsoft ومصادر وثائق الـ Win32 المتنوعة .
- **EIntError**: class اساسية لـ exceptions لحساب العدد الصحيح . لا يتم انشـاءها ابداً كـ exception object ، ولكن يمكن استـخدامـها للايقاع بكل خطأ عمليات الاعداد الصحيحة . لانشاء exception objects لاعداد صحيحة معينة ، اســتـخــدم بدلاً من هذه الـ exception classes المشـتـقـة EDivByZero ، ERangeError، و EIntOverflow .
- **EIntfCastError**: يحدث عندما تحاول عبارة ان تصنع class غير مناسبة مع عامل الـ as .
- **EIntOverflow**: قيمة العدد الصحيح أكبر من مداه المحدد .
- **EInvalidArgument**: قيمة خارج النطاق فى function محاسبة مالية فى وحدة الـ Math.
- **EInvalidCast**: تغيير type-cast غير صحيح .



- **EInvalidGraphic**: يحدث عندما يحاول البرنامج ان يفتح ملف جرافيك غير معروف (مثلاً، يفتح عن طريق الخطأ ملف نص كـ bitmap).

- **EInvalidGraphicOperation**: يشير الى عملية جرافيك غير مسموح بها مثل محاولة تغيير حجم ايقونة.

- **EInvalidGridOperation**: يحدث لعمليات الـ grid الغير مسموح بها مثل محاولة الوصول الى خلية غير موجودة.

- **EInvalidOp**: عملية برنامج معالجة حسابية لـ floating-point غير صحيحة. هذا الـ exception يحدث للاخطاء الرياضية الغير معرفة عندما، مثلاً، تكتشف الـ CPU أمر غير معرف، تحاول اداء عملية غير صحيحة.

- **EInvalidOperation**: يشير الى عملية غير صحيحة في component، وبخاصة واحدة تتطلب window handle لـ component تكون خاصية الـ Parent له غير معينة. يمكن ان يحدث ايضاً لعمليات السحب والاسقاط الغير مسموح بها.

- **EInvalidPointer**: pointer غير صحيح، مثلاً، محاولة استخدام nil pointer أو ترك block ذاكرة مخصصة أكثر من مرة.

- **EMathError**: class اساسية لـ exceptions حساب الـ floating-point. لا تستخدمه مباشرة كـ exception object استخدم بدلاً منه exception classes المشتقة من EInvalidArgument، EInvalidOp، EOverflow، EUnderflow، وEZeroDivide.

- **EMCIDeviceError**: يحدث لمشكلات مع الـ Media Control Interface (MCI).

- **EMenuError**: يشير الى عملية غير مسموح بها في الـ menu.

- **ENoResultSet**: يحدث بواسطة objects قاعدة بيانات الـ Query المستخدمة بدون اشتراط عبارة SELECT.

- **EOleCtrlError**؛ يحدث عندما تحاول Ole فاشلة ان ترتبط بـ ActiveX control.
- **EOleException**؛ يحـدث عند أى فـشـل مع اسـتـدعـاء الـ IDispatch method.
- **EOleSysError**؛ يحدث اذا فشل method استدعاء الـ IDispatch .

* EOutOfMemory : عدم القدرة على تحديد أماكن الذاكرة المطلوبة .

- **EOutOfResources**؛ يحـدث عـند حـدوث مـحـاولـة فـاشلـة لانشـاء Windows handle.
- **EOverflow**؛ عدم تكون قيمة الـ floating-point أكثر من المدى المسموح به .
- **EPackageError**؛يحدث لأى اخطاء تشمل الـ packages .
- **EPrinter**؛ يشير الى خطأ يحدث اثناء الطباعة .
- **EPrivilege**؛ يحدث لاخطاء processor privilege مثل محاولة تنفيذ أمر غير مسموح فى مستوى الـprivilege الحالى .
- **EPropertyError**؛ يحدث اذا لم يكن ممكناً تحديد قيمة خاصية ، على سبيل المثال، اذا كانت خارج النطاق أو نوع بيانات غير مسموح به .
- **EPropReadOnly**؛ يشير الى محاولة غير مسموح بها لكتابة خاصية قراءة فقط باستخدام الـ OLE .
- **EPropWriteOnly**؛ يشير الى محاولة غير مسموح بها لقراءة من خاصية كتابة فقط باستخدام الـ OLE .
- **ERangeError**؛ قـيـمـة خـارج النطاق لعـمليـة فـهـرسـة array أو short-string، وكذلك للتعيينات لمتغيرات النطاق الفرعى . هذا الـ exception لا يتم حـدوثه للـ long strings . انه يتطلب ان يكون البـرنامج قـد حـدث له compiled مع مفتـاح الـ {$R+} ، وهذا مـا يحـدث فـقط اثنـاء ازالة الاخطاء . ان



التحديد الافتراضى {$R+} يغلق فحص المدى، ويمنع هذا النوع من الـexception من ان يحدث .

- **EReadError**: يحدث عند محاولة خاطئة لقراءة بيانات خطء، وايضاً اذا كان الـ Delphi غير قادر على قراءة قيمة خاصية اثناء انشاء form object .

- **ERegistryException**: يشير الى فشل لعملية تم أداءها على الـ Windows registry مثل محاولة تعديل مساحة من السجل مع امتيازات مستخدم غير كافية .

- **EResNotFound**: يحدث اذا كان resource معين مثل الايقونة غير موجود. والعنصر الاساسى فى هذه المشكلة هو امر {$R *.DFM} الذى تم حذفه خطأ فى قطاع الـ implementation فى source code module للـ form unit .

- **ESocketError**: يشير الى فشل مع الـ Windows socket object. .

* EStackOverflow : يحدث عندما تنمو الـthread stack الحالية فى "final guard page,"، مما يشير الى ان البرنامج أوشك على على ان يفتقر الى الذاكرة والاسباب لهذه المشكلة هى المتغيرات المحلية الكبيرة جداً فى الـ procedures و functions وكذلك فى النظم الفرعية المتكررة المتداخلة. (انقل المتغيرات الكبيرة خارج الـ procedures والـ functions .

- **EStreamError**: يشير الى خطأ مع عملية streaming مثل تحديد index غير صحيح لـ list-box .

- **EThread**: يشير الى مشكلة تزامن الـ thread .

- **ETreeViewError**: يحدث عندما يتم استخدام index غير مسموح به مع TreeView .

- **EUnderflow**: عندما تكون قيمة العمليات للـfloating-point أقل من المدى المسموح به

- **EVariantError**: يشير الى خطأ مع نوع بيانات المتغير مثل type-cast غير صحيح، أو index خارج المدى .

- **EWin32Error**: يتم حدوث هذا exception اذا تم اكتشاف خطأ Windows . يعرض برنامج معالجة الـ exception الافتراضى dialog يذكر قيمة الـ code الخاطئة ورساله تحديد الخطأ. يمكنك استخدام الـ SysUtils Win32Check function مع هذا الـ exception لتبحث عن الرساله من نظام التشغيل .
- **EWriteError**: يحدث لاخطاء كتابة الـ file-stream .
- **EZeroDivide**: Floating-point تحاول القسمة على صفر .

البحث عن خصائص Exception Class:

ابحث عن انواع معينة من exceptions فى الـ online help الخاصة بـ Delphi. ثم ابحث عن التعريف الخاصة لها. هذا يخبرك عن نوع- الخصائص التى تعرفها الـ exception class. على سبيل المثال، تقوم الـ SysUtils unit بتعريف الـ EInOutError كما يلى:

```
EInOutError = class(Exception)
 public
  ErrorCode: Integer;
 end;
```

هذا التعريف يخبرك ان، بالإضافة الى خاصية الـ Message التى توفرها كل الـ exceptions، يوفر الـ EInOutError ايضا حقل ErrorCode. ان ملف الـ code للـ Sysutils.pas source موضوع فى دليل تركيب Delphi وهو Source\Rtl\Sys. يتم تعريف exception classes معينة فى كثير من ملفات الـ source code الاخرى لـDelphi.

ويمكنك ايضاً استخدام هذه الـclass لأغراض التعريف الخالصة دون انشاء أشارة للـ exception instance. على سبيل المثال، قم باستبدال عبارة الـ on-do فى القائمة (١٩-١١) بما يلى:

```
on EDivByZero do
  ShowMessage('Divide error');
```

يقارن السطر الاول الـclass exception instance بـ EDivByZero، واذا كان الـ object من هذه الـclass يستدعى الـ ShowMessage وتنتقل الـ

exception من الانواع الاخرى الى أعلى من خلال استدعاء السلسلة . استخدم هذه التقنية اذا لم تكن تحتاج الى الاشارة الى الـ exception instance أو استخدم هذه الـclass :

```
on E: EDivByZero do
 ShowMessage(E.Message);
```

لعمل أشارة لـ exception instance E وعرض message string المحتوى فيها .

معالجة الـ exceptions المتعددة:

ان الـ except block قد يستجيب لأكثر من نوع من الـ exception . على سبيل المثال ، توضح جزئية الـclass التالية كيف معالجة ثلاثة انواع من الـ exception object ؛ تنتقل الانواع الاخرى لأعلى السلسلة حتى تجد معالم :

```
on EDivByZero do
 ...;
on EInOutError do
 ...;
on EOutOfMemory do
 ...;
```

يمكنك ايضاً تعريف exception instances للتوصل الى معلومات اخرى متنوعة متوفرة مع أنواع exception معينة . على سبيل المثال ، تمسك جزئية الـ code التالية بثلاثة انواع من الـ exception ، وتعرف exception instancesE لكل نوع :

```
try
 {Statements that might raise one of the following
  exceptions}
except
 on E: EDivByZero do
    ShowMessage(E.Message + ' (I = 0)');
     on E: EInOutError do
    with E do
       ShowMessage(Message + ' #:' + IntToStr
```

```
            (ErrorCode));
              on E: EOutOfMemory do
        begin
       ShowMessage(E.Message);
    // free reserved memory here
     end;
    end;
```

ان كل exception handler في الجزئية السابقة يستخدم المعلومات الموجودة في الـ E بطريقة مختلفة. يمسك المعالج الاول خطأ قسمة عدد صحيح على صفر، ويعرض خاصية الـ Message للـ E مع string حرفي ملحق بها يوفر قليلاً من المعلومات عما جرى من خطأ.

ويمسك الـ exception handler الثاني باخطاء الـ I/O. إنه يستخدم عبارة with للتوصل لخصائص الـ E، ويعرض الـ E.Message string مع متغير الـ E.Errorcode وقد تم تعريفه للـ EInOutError exception class. يتم تحويل هذه القيمة الى string مع استدعاء IntToStr function الخاصة بـ Delphi.

ويمسك الـ exception handler الثالث باخطاء الـ out-of-memory. وهو يعرض الـ E.Message. ويشير التعليق الى مكان جيد لتحرير قالب الذاكرة المحجوز والذي قد يخصصه البرنامج لتوفير بعض الذاكرة الاضافية، فقط لاغراض السماح للمستخدم بالخروج من البرنامج، أو لانهاء العملية التي تسببت في ظرف الـ out-of-memory.

تحذير: عندما تستخدم الـ الـ on-do لمعالجة exception classes معينة، فإن وجود exception اخرى غير معالجة تجعل الـ procedure أو function الحالية تنتهي على الفور. في مثل هذه الحالات، استخدم الـ try-finally block لتحرير الذاكرة المخصصة، لإغلاق الملفات، ولأداء عمليات اخرى واجبة الفعل.

حدوث exception جديد:

يتم حدوث العديد من الـ exceptions بصورة تلقائية، على سبيل المثال خطأ القسمة على صفر، ولكن يمكنك ايضاً ان تحدث exception خاصة بك لتقرر اخطاء أو حالات غير عادية.

استخدم الكلمة الاساسية raise عند حدوث exception جديد . بعد الـ raise، قم بانشاء exception instance من الـ Exception class أو أى class مشتقة . بافتراض ان البرنامج له متغير الـ Boolean وهو ErrorFlag الذى ، اذا كان True ، يشير الى مشكلة ، قد يحدث البرنامج exception بـ code مثل هذه :

```
if ErrorFlag then
  raise Exception.Create('Error');
```

وبدلاً من الـ Exception ، يمكنك استخدام اغلب الـ classes المذكورة سالفاً . يمكنك ايضاً حدوث exceptions من classes خاصة بك ولكن سأوضح المزيد عن هذا فيما بعد . وتأتى نتائج هذه الـ code مطابقة للـ exceptions التى تحدثها الـ components مثل أخطاء الآلة ، أو أى مصدر آخر . يمكنك كتابة برامج معالجة exception للـ exception التى تحدثها الـ code ، أو يمكنك ان تجعل برامج معالجة الـ exception الاصلية تتولى امرهم .

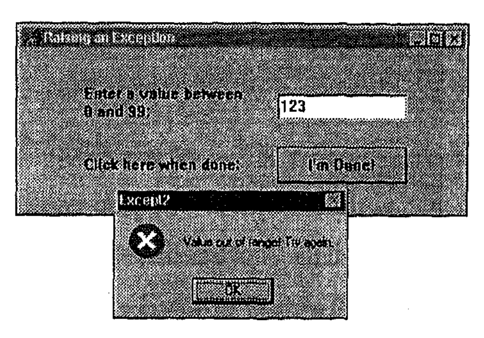
ملحوظة: ان حدوث exception يبحث عن الفور عن اقرب exception فى الـ procedure أو الـ function التى تستدعى الـ raise . اذا لم يكن لـ procedure أو function أية معالجة ، تنهى الـ raise بفاعلية الـ subroutine .

تحذير: كمثال على حدوث الـ exceptions ، قم بتشغيل برنامج العرض Except2 على القرص المدمج فى دليل Source\Except2 ، ثم ادخل قيمة من صفر الى تسعة وتسعين فى نافذة الـ Edit . يوضح شكل (١٩-٨) عرض البرنامج . اضغط زر الـ I'm Done لانهاء البرنامج . قم بتشغيل البرنامج مرة اخرى وفى هذه المرة داخل قيمة غير مسموح بها مثل ١٠٠ . كما ترى ، هذا يؤدى الى حدوث exception يمنع البرنامج من الاغلاق . ان برامج معالجة الـ exception الافتراضية لـ Delphi تعنى بتعريف المشكلة . توضح القائمة (١٩-١٢) OnClick للزر .

القائمة (١٩-١٢): Except2\Main.pas للـ Button

```
procedure TMainForm.Button1Click(Sender: TObject);
var
  N: Integer;
```

```
begin
 N := StrToInt(Edit1.Text);
   if (N < 0) or (N > 99) then
  raise ERangeError.Create('Value out of range! Try
     again')
      else begin
     ShowMessage('Success! Click Ok to end program');
       Close;
      end;
end;
```



شكل (١٩-٨): إدخال قيمة خارجة عن المدى فى نافذة الـ Edit لـ Except2 يؤدى الى حدوث exception يمنع البرنامج من الانتهاء

فى الـ Button OnClick، تحول الـ StrToInt خاصية الـ Text التابعة للـ Edit1 الى قيمة عدد صحيح، وتعين هذه القيمة بـ N وترفع عبارة الـ if التالية exception اذا كانت الـ N ليست داخل المدى من صفر الى ٩٩ . بعد الكلمة الاساسية raise، ينشئ البرنامج exception instance من الـ ERangeError class. فى هذا البرنامج، يتلقى برنامج معالجة الـ exception الافتراضى الـ ERangeError ويعرض الرسالة فى الـ dialog box.

ومن الجدير بالذكر ان الـ event handler قد يفشل بسبب نوع آخر من الـ exception: خطأ تحويل تم حدوثه بواسطة استدعاء الـ StrToInt. قم بتشغيل البرنامج مرة اخرى وادخل 3.14159 أو اسمك فى نافذة الـ Edit، ثم اضغط الزر فى هذه المرة ترى رسالة مختلفة مثل (3.14159 is not a valid Integer value) أو الرقم 3.14159 ليس قيمة صحيحة صالحة يتم عرض هذه الرسالة

بواسطة برنامج معالجة الـ exception الافتراضى لـ Delphi. والاهم من هذا ان الـ exception الذى يتم حدوثه بواسطة الـ StrToInt يجعل الـevent handler ينتهى على الفور، مما يلغى باقى الـ procedure code.

On the CD-ROM

على القرص المدمج: وهناك نسخة افضل للـevent handler هذا تستخدم الـ try-except block لمعالجة هذا النوع من المواقف، وتعطى المستخدم فرصة ليصحح الادخال الخاطئ، بغض النظر عن السبب. قد تكون لاحظت انه عليك ان تضغط الـ Tab أو تنقر الفأرة لتعود الى نافذة الـ Edit بعد اغلاق message الخطأ. توضح القائمة (١٩-٣) OnClick المأخوذة من مشروع الـExcept3 على القرص المدمج فى دليل الـ Source\Except3.

يستخدم الـ procedure الذى تم مراجعته procedure متداخلاً لعرض رسالة خطأ بالإضافة إلى الجملة Try again. ويستدعى ايضاً الـ SetFocus method للـ Edit1 لكى يجعل من السهل على المستخدمين ان يدخلوا قيمة اخرى بعد قراءة رسالة الخطأ. ان لب الـ procedure يحدث exception اذا كانت الـ N خارج المدى. وإنه يستخدم ايضاً الـ try block بحيث ان الانواع الاخرى من الـ exception كتلك التى يمكن حدوثها باستدعاء الـ StrToInt يتم معالجتها هى الاخرى. ويوضح الـexcept block كيفية معالجة نوعين من المشاكل:

اخطاء التحويل التى يتم حدوثها بواسطةالـ StrToInt، واخطاء العدد الصحيح التى يتم حدوثها بواسطة الـ procedure ذاته. فى كلتا الحالتين استدع الـHandler المتداخل لعرض الرسالة وحدد الـ focus مرة اخرى الى الـEdit. وأى انواع اخرى من المشكلات يتم تمريرها الى برامج معالجة لـ exception الافتراضية لـ Delphi.

القائمة (١٩-١٢): Except3\Main.pas (OnClick للـButton)

```
procedure TMainForm.Button1Click(Sender: TObject);
var
  N: Integer; S: String;
    // Nested exception handler
    procedure Handler(Message: String);
    begin
   ShowMessage(Message + ' Try again.');
```

```
          Edit1.SetFocus;
            end;
  begin
    try
          N := StrToInt(Edit1.Text);
            if (N < 0) or (N > 99) then
              raise ERangeError.Create('Value out of range!')
                else begin
              ShowMessage('Success! Click Ok to end
            program');
                  Close;
                end;
    except
      on E: EIntError do Handler(E.Message);
            on E: EConvertError do Handler(E.Message);
    end;
  end;
```

Tip **فكرة:** لا يجب ان يكون الـ Handler procedure مـتـداخـلاً فى procedure آخـر فـيـمكن ان يتم تعـريفـه كـ procedure . أو class method يمكنك ان تفـعل هذا، مـثـلاً، بحيـث تسـتطيع الـprocedure المتعددة استدعاء نفس البرنامج المعالج.

إعادة حدوث الـ exception:

بينـمـا تقـوم بتطوير تطبـيـقـك، فـإنك تكتب العـديـد من الـ procedure والـ functions التى توفر برامج معـالجة exception، وقـد تريد ان تضيـف امكانات جديدة لهذا الـ code الموجـودة فعلاً. على سبيل المثال، قد يكون لديك برنامجاً يغلق ملفاً أثناء الحدوث لخطأ قرص. وقد تحتاج تركيبة اخرى لمعالجة نفس هذا الـ exception لاغراض خـاصة بها. فـبدلاً من اغلاق الملف فى مكانين مـخـتلفين- وهى عـمل برمـجى ضـعـيـف يعـقـد الامـور فى المسـتـقـبل- فـيـسـتطيع مـعـالج الـ exception الثانوى ان يعيد حدوث الـ exception، هذا يحافظ على exception بحيث، بعد ان يؤدى البرنامج المعالج الثانوى اعـماله، ينتقل الـ exception الى أعلى سلسلة الاستدعاء ليصل الى برامج معالجة اخرى تكون فى انتظارة.

اذا كنت على دراية بالبرمجة المختصة بالـ objects فقد تفكر فى إعادة exception كنوع من التقسيم الفرعى للـ class فى وقت التشغيل. فقد يزيد البرنامج المعالج للـ exception على البرامج المعالجة الاخرى بالامساك بأنواع معينة من الـ exception، لتنفيذ بعض الاعمال، ومن ثم اعادة حدوث الـexception للابقاء عليه لمزيد من المعالجة. هذا يشبه الطريقة التى يستدعى بها method ما classes مشتقة method موروثاً ليزيد مما تفعله هذا الـ code.

لمعالجة exception دون أن تدمر الـ exception instance استخدم الكلمة الاساسية raise وحدها دونargument. افعل هذا مع كل انواع الـ exception باستخدام code مثل التالية (يوضح التعليق اين تضع عبارة واحدة أو أكثر والتى قد تحدث الـ exception الاصلى).

```
try
  { Statements that might raise an exception }
except
  ShowMessage('OOPs!');
    raise;  { Reraise the exception }
end;
```

اذا قامت أى عبارة فى الـtry blockبإحداث أى exception يقفز التنفيذ الى الـShowMessage. بعد ذلك، فإن استخدام الـ raise وبدون argument يعيد حدوث الـ exception. ان تشغيل هذا الـ code يعرض رسالتين: الاولى بواسطة الـShowMessage، والثانية بواسطة برنامج معالجة الـ exception الافتراضى للبرنامج (ولكن، قد تكون بعض الـ exception ولا تعرض أية رسائل-المزيد عن هذا فيما بعد).

تحذير: تقوم الـ code السابقة بالامساك بكل الـ exception وهو أمر غير معقول بوجه عام-وقد يؤدى الى مشكلة كبيرة-مثلاً، عدم معالجة حالة الخروج عن الذاكرة بشكل سليم. ولكن، لان هذا الـ code ايضاً يعيد حدوث كل الـ exception، فإن برامج معالجة الـ exception الافتراضية لـ Delphi مازال لديها فرصة ان تتولى امر المشكلات الخطيرة. وفى حالات نادرة فقط عندما يكون لديك اسباباً جيدة لخرق هذه القاعدة، استدع الـ raise دائماً كما هو موضح فى الـ except block الذى يمسك بكل الـ exception.

لإعادة حدوث نوع معين من الـ exception استخدم code مشابهة، ولكن إدخل raise. على سبيل المثال، فإن جزئية هذا الـraise:

```
try
{ Statements that might raise an exception }
except
  on E: EOverflow do
     begin
       ShowMessage('OOPs!');
         raise;  { Reraise specific exception }
       end;
end;
```

تمسك الحالات من الـ EOverflow class، وتعرض رسالة، ثم تعيد حدوث نفس الـ exception. و الـ exception الاخرى التى تحدث فى عبارات الـtry block تنهى الـprocedure أو الـ function على الفور، متخطية الـ except block.

ملحوظة: حدوث exception ينهى على الفور الـ procedure أو الـ function التى تستدعى الـ raise.

إنشاء Exception Classes:

عندما تحتاج الى حدوث exception، يمكنك اختيار واحدة من Delphi classes المشتقة من الـ Exception (راجع القائمة السابقة) أو يمكنك انشاء exception classes الخاصة بك. يمكنك ان تشتق classes الخاصة بك من الـ Exception، أو يمكنك ان تنشئ الـ Exception.

قم باشتقاق exception classes الخاصة بك من الـ Exception اذا كنت تريد ان تقوم برامج المعالجة الافتراضية للتطبيق بالامساك بأى exception غير معالجة لـ class. وبشكل عام، هذا هو افضل شئ تفعله. ولكن، يمكنك إنشاء class جديدة من تصميمك اذا لم تكن تريد لبرامج المعالجة الافتراضية ان تتعرف على exception.



تحذير: ان أى exception غير معالجة للـ classes ليست مشتقة من الـ Exception التى تؤدى الى خطأ تطبيق جسيم . يجب ان تعالج الـ cade كل حدوث لـ exception غير مستندة على الـ Exception class .

:Custom exception classes

ان custom exception classes تعتبر نافعة فى تجميع معلومات حول مشكلات تحدث فى برنامجك . على سبيل المثال ، ان procedure رياضى يؤدى بعض انواع الحسابات قد يمرر قيم متغيرات غير مسموح بها فى شكل من custom exception class . فى الـ except block للبرنامج ، تستطيع عبارة ما ان تعرض هذه القيم .

هذا يعطى مستخدمى برنامجك معلومات يستطيعون استخدامها لاصلاح مشكلة-ادخال قيم صحيحة فى الـ display ، مثلاً . ولا احد يقدر رسالة خطأ مثل "File I/O error." . اما رسالة مثل . "Filename contains the illegal character @" فتعطى مزيداً من المعلومات . يمكنك استخدام الـ exception custom class لانشاء هذا النوع من رسالة الخطأ . وتعتبر الـ customexception class غاية النفع ايضاً فى ازالة الاخطاء .

ان ابسط mothed لانشاء custom exception class هو بتحديد نوع classجديدة مستندة على الـ Exception clas . على سبيل المثال ، إدخل تعريف الـ type التالى (يمكن ان تذهب فى التنفيذ ، ولكنك غالباً ما تريد ان تجعل الـ class متاحة لـ modules اخرى بحيث يستطيعوا انشاء ومعالجة exception للـ class الجديدة) :

```
type
  TCustomException = class(Exception);
```

توفر الـ class الجديدة اسم class فريد لتعريف نوع معين من المشكلات-لا تحتاج الـ class الى هيكل . مع هذا التعريف ، يمكنك احداث exception للـ TCustomException class ، وتمسك بالاخطاء الخاصة بهذا النوع المعين . على سبيل المثال ، يمكنك ادخال هذه العبارة فى الـ Button OnClick :

```
raise TCustomException.Create('Custom exception');
```

عندما تقوم بتشغيل البرنامج وضغط الزر، تتلقى برامج معالجة الـ exception الافتراضية لـ Delphi حالة الـ exception وتعرض الرسالة. وتستطيع ايضاً برامج الـ exception الخاصة بك ان تمسك الـ TCustomException objects باستخدام code مثل هذه:

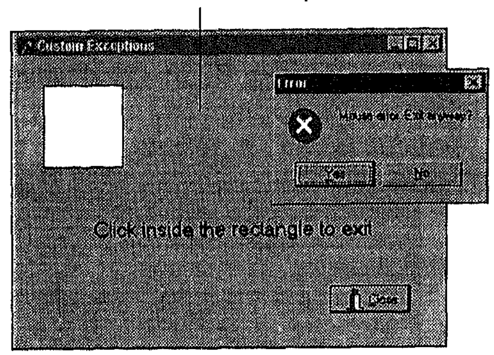
```
try
   { Statements that might raise an exception }
except
  on E: TCustomException do
      ShowMessage(E.Message);
end;
```

هذا يشبه الامثلة الاخرى الموجودة فى هذا الباب، والفرق الوحيد هو ان الـ code تمسك بالـ TCustomException class exception فقط والانواع الاخرى من الـ exception تمر الى أعلى فى سلسلة الاستدعاء حتى تجد معالج أو تصل الى برامج معالجة الـ exception الافتراضية لـ Delphi.

واحد الاسباب لانشاء الـ exception class الخاصة بك هو لتخزين قيم توفر معلومات إضافية حول خطأ ما. على سبيل المثال، افترض انك تريد ان يقوم البرنامج مباكتشاف ما اذا كان المستخدمون قد ضغطوا داخل تكوين جرافيك يقوم البرنامج برسمه. ولان التكوين لا يعد exception، فيجب ان تفحص كل ضغط الفأرة فى نافذة الـ form وتستجيب بشكل ملائم- تنهى التطبيق، مثلاً، فقط اذا كان المستخدم قد ضغط داخل تكوين معين. تستطيع بالطبع ان تقوم بهذا باستخدام Delphi components، ولكن هذه المشكلة توضح الاستخدام الجيد للـ custom exceptions.

On the CD-ROM

على القرص المدمج: يوضح تطبيق الـ MouseExcept على القرص المدمج فى دليل Source\MouseExcept كيفية إنشاء استخدام exception. يوضح شكل (١٩-٩) عرض البرنامج توضح القائمة (١٩-٤) الـ source code للبرنامج.



شكل (١٩-٩): يوضح تطبيق الـMouseExcept إنشاء واستخدام exception مخصصاً. اضغط داخل المربع الصغير لانهاء البرنامج؛ اضغط فى أى مكان آخر فى النافذة لحدوث exception

القائمة (١٩-٤): MouseExcept\Main.pas

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics,
    Controls,
      Forms, Dialogs, StdCtrls, Buttons, ExtCtrls;

type
  TMainForm = class(TForm)
      BitBtn1: TBitBtn;
        Label1: TLabel;
        procedure FormMouseDown(Sender: TObject;
          Button: TMouseButton; Shift: TShiftState; X, Y:
      Integer);
        procedure FormPaint(Sender: TObject);
      private
        { Private declarations }
        procedure CheckMouseLocation(X, Y: Integer);
        procedure ExitOnClick(X, Y: Integer);
```

```
    public
    { Public declarations }
        end;

var
 MainForm: TMainForm;

implementation

{$R *.DFM}

const
 rLeft   = 25;
    rTop    = 25;
    rRight  = 100;
    rBottom = 100;

type
 TMouseException = class(Exception)
      X, Y: Integer;
         constructor Create(const Msg: string; XX, YY:
      Integer);
 end;

constructor TMouseException.Create(const Msg: string;
 XX, YY: Integer);
begin
 X := XX;   // Save X and Y values in object
    Y := YY;
   Message :=  // Create message string
     Msg + ' (X=' + IntToStr(X) + ', Y=' + IntToStr(Y) +
     ')';
end;

  procedure TMainForm.CheckMouseLocation(X, Y:
  Integer);
```

```
    begin
     if (X < rLeft) or (X > rRight) or
          (Y < rTop)  or (Y > rBottom) then
            raise
           TMouseException.Create('Mouse location error',
          X, Y);
end;

procedure TMainForm.ExitOnClick(X, Y: Integer);
begin
  try
      CheckMouseLocation(X, Y);  // Bad values raise
      exception
        Close;                   // Exit the program
       except
      on TMouseException do
        begin
     if MessageDlg('Mouse error. Exit anyway?',
               mtError, [mbYes, mbNo], 0) = mrYes
              then
      Close
              else
             raise;
             end;
         end;
end;

procedure TMainForm.FormMouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  ExitOnClick(X, Y);
end;

procedure TMainForm.FormPaint(Sender: TObject);
begin
  Canvas.Rectangle(rLeft, rTop, rRight, rBottom);
```

```
end;

end.
```

ان الـ form OnPaint الرئيـسـيـة لبـرنـامج الـ MouseExcept يرسم مـربع صغير باستخدام قيم ثابتة معرفة فى قطاع الـ implementation عندما تضغط الفـــأرة فى أى مكان فى الـنافـــذة، يقـــوم الـ FormMouseDown للـ OnMouseDown بتلقى قيم إحداثيات الـ x والـ y الخاصة بمؤشر الفأرة. وهذا الـ procedure يستدعى الـ ExitOnClick، محرراً نفس هذه القيم x و y على انها arguments. إن الـ ExitOnClick يعد public procedure قمت بإضافته للـ form class وقد تم تنفيذه كما هو موضح فى القائمة.

ينفـذ الـ ExitOnClick اثنين من العبـارات فى الـ try block. اولاً، يقـوم باســـتـــدعـــاء ثانى procedure مـــضـــافـــاً الى الـ class، وهـو الـ CheckMouseLocation، والذى يختبـر ما اذا كـانت الـ x والـ y داخل المربع. ويقـوم الـ procedure ايضـاً باسـتـدعـاء الـ Close لانهـاء البـرنامج. ولان الـ CheckMouseLocation يحـدث exception اذا مـا تم إكتشـاف أى مـشكلات خاصة بالـ x والـ y، فإن عبارة الـ Close يتم تخطيطها اذا ماضغط المستخدم خارج المربع. وفى هذا الـ event، يقـــوم الـ except block الخـــاص بالـ exception بعرض رسالة خطأ ويسأل عمـا اذا كان ينهى البرنامج بأية حال. لقد قمت ببرمجة الـprocedure بهـذه الطريقـة بحيث، اذا مـا اجبت بـ No على dialog الرسـالة، يقـوم الـ except block باعـادة حـدوث الـ exception. لذلك يوضح البـرنامج الاختيـارى كلاً من استجابة البرنامج للـ exception المخصص ويوضح ما يحدث عندمـا تتلقى برامج مـعـالجـة الـ exception الافـتـراضـيـة لـ Delphi نفس هذا الـobject.

ويتم تعريف هذا الـ object على انه حالة من الـ class يعرفها البرنامج نفسه كما يلى:

```
type
  TMouseException = class(Exception)
    X, Y: Integer;
```

```
constructor Create(const Msg: string; XX, YY:
Integer);
  end;
```

تعتبر الـ TMouseException مشتقة من الـ Exception class التابعة لـ Delphi. وبالإضافة الى العناصر الموروثة عن الـException، فإن الـ TMouseException تضيف اعضاء من اعداد صحيحة x و y، وتعرف constructor، والذى يتم استدعاءه لإنشاء class objects، ومحتويات الـ constructor كمايلى :

```
constructor TMouseException.Create(const Msg: string;
 XX, YY: Integer);
begin
 X := XX;    // Save X and Y values in object
   Y := YY;
   Message :=  // Create message string
  Msg + ' (X=' + IntToStr(X) + ', Y=' + IntToStr(Y) + ')';
end;
```

ان اول عبارتين فى الـ constructor يحفظ قيم parameters الـ XX والـ YY فى الـ x والـ y المعرفين فى الـ TMouseException. ويتم تعيين رسالة تتضمن هاتين القيمتين للـ Message string الموروث عن الـ Exception.

ولكى تفهم كيف يتم استخدام الـ exception class المخصص، انظر الـ CheckMouseLocation procedure. اذا كانت الـ X أوالـ Y خارج المربع الذى يرسمه البرنامج، فإن هذا procedure يحدث للـ exception object TMouseException class مستخدماً هذه العبارة :

```
.raise

  TMouseException.Create('Mouse location error', X, Y);
```

هذا يؤدى الى إنشاء TMouseException، والذى يحفظ قيم الـ X والـ Y لموقع المؤشر المتحرك، وينهى الـ CheckMouseLocation على الفور. وتعود الى الـ ExitOnClick، اذا حدث هذا الـ exception، يتم تخطى عبارة الـ Close ويتولى الـ except block التحكم.

بالرغم من ان هذا البرنامج النموذجى قد يبدو معقداً للغاية بحيث يصعب أداء عملياته، فإن فى تطبيق أكثر اتساعاً ذا صور جرافيكية كثيرة، تقوم مثل الـ TMouseException class بتبسيط الـ code الى حد كبير. وبدون الـ exception class المخصصة، فقد يستخدم البرنامج العديد من عبارات if الشرطية التى يصعب متابعتها لاختبار مواقع الفأرة، مما يجعل إزالة الاخطاء امراً عسيراً. ان تركيز كل معالجة الاخطاء فى exception class مخصصة، وباستخدام الـtry-except blocks لاكتشاف الاخطاء، يحافظ على الـ code ويحسن من قدرتها.

exception classes الغير مشتقة:

يمكنك ايضاً انشاء exception classes جديدة تماماً غير مستندة على class الـ Exception. وابسط طريقة class ليس لها هيكل. ولكن لأنها Class اساسية فيجب ان ينتهى تعريفها بالكلمة الاساسية end وفصلة منقوطة. وهذا هو احد الأماكن النادرة فى الـObject Pascal تكون فيه الـ end غير مطلوبة ليكون هناك body. على سبيل المثال، ابدأ تطبيقاً جديداً وإدخل تعريف الـ type هذا فى قطاع الـ implementation:

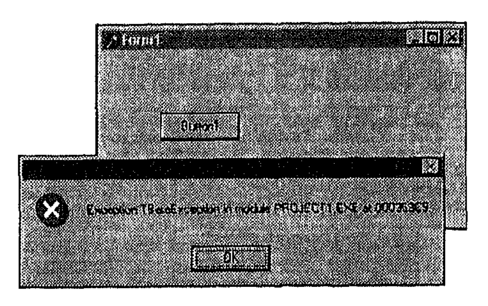
```
type
  TBareException = class
    end;
```

بعد ذلك، أضف Button على الـ form، وقم بانشاء OnClick خالى ويضغط هذا الـ event مرتين فى نافذة الـ Object Inspector. قم ببرمجة الـ event handler كما يلى:

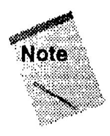
```
procedure TForm1.Button1Click(Sender: TObject);
begin
  raise TBareException.Create;
end;
```

ان استدعاء الـ Create method الخاص بالـ claas يؤدى الى إنشاء object من الـ class، والذى يتم حدوثه كـ exception. عندما تقوم بالـcompile وتشغيل البرنامج بضغط F9، فإن نقر الزر ينفذ هذا الـ code ويحدث الـ

exception . ويتلقى الـ exception handler الافتـراضى لـ Delphi فى هذا الـ object، ويعرض الـ message dialog الموجود فى شكل (١٩-١٠).



شكل (١٩-١٠): exception handler الافتراضى لـ Delphi يعرض هذه الرسالة لـ exception object من class ليست مشتقة من الـ Exception



ملحوظة: لان كل الـ class تكون مشتقة من الـTObject، فحتى الـ class الاساسية الجديدة تأتى مع الـ Create والـ Free methods المقامة بالفعل .

اذا استخدمت هذه التقنية، فمن الافضل ان تعالج كل حالات الـ exception classes. الخاصة بك، وقد تريد ايضاً ان تقوم بتثبيت برنامج معالجة الـ exception الافتراضى الخاص بك كما هو موضح مؤخراً فى هذا الباب . يمكن للـ class ان تعرف methods ومتغيرات اضافية وما تصنعه فى exception classes. المخصصة التابعة لك أمر يرجع إليك .

:Exception base class

يجعل الـ exception classes الخاصة بك ترث الـ Exception classes، فإن الـ classes ترث ثروة من الاعضاء . على سبيل المثال، تقدم الـ Exception class عدد من الـ methods لإنشاء exception class بطرق مختلفة . وكل هؤلاء الـ constructors يحملون اسماء تبدأ بالـ Create، وكلهم لديهم نفس الهدف الوحيد وهو إنشاء string رسالة خطأ من انواع مختلفة من الـ arguments . يمكنك استدعاء هؤلاءconstructorsلإنشاء exception instances من الـ class المشتقة . أو يمكنك استدعائهم لـException class أو لأى من الـ classes المشتقة المذكورة آنفاً فى هذا الباب . سوف أمر على(constructors) التابعين لـ class Exception باختصار هنا-ومن المؤكد ان يفى احدهم باحتياجاتك .

وابسط هؤلاء هو الـ Create الذى رأيته بالفعل. قم بتمرير ثابت أو متغير string كرسالة خطأ:

```
raise Exception.Create('Trouble in Paradise');
```

Tip 

فكرة: ادخل العبارة السابقة فى الـ OnClick Button، واضغط الزر لترى النتائج. افعل نفس الشئ مع الامثلة الاخرى المذكورة فى نفس هذا الفصل.

لإنشاء رسالة تحتوى على معلومات إضافية، قيم متغيرة، مثلاً، استدع constructor البديل CreateFmt. والى (constructors)، قم بتمرير string مع اوامر معينة زائد مجموعة من القيم ليتم ادخالها فى الـ string. (للمعاونة فى هذه التقنيات انظر (استخدام دالة الـ Format فى الباب السابع، وكذلك انظر الـ Format فى الـ Delphi online help). على سبيل المثال، تقوم هذه العبارة بإحداث exception مع رسالة خطأ تعرض إثنين من القيم الصحيحة:

```
raise Exception.CreateFmt('Error: X=%d Y=%d', [X, Y]);
```

ونتيجة لهذه العبارة، يرى المستخدمون رسالة خطأ مثل التالية:

```
Error: X=-1 Y=12
```

عند انشاء exception objects بهذه الطريقة، فقد تريد استخدام. مجموعة error message resource strings فى الـ SysConst unit الخاصة بـDelphi. يذكر الجدول (١٩-١) الثوابت الموجودة فى هذا الجدول ويوضح الـ strings المرتبطة بها. إنها متصلة بكل التطبيقات، لذا يمكنك استخدامها بدلاً من تغريف الـ strings الشبيهه الخاصة بك لاستخدام الـ strings اضف تعريف الـ uses الى قطاع. الـunit implementation:

```
uses
  SysConst;
```

بعد ذلك، قم بإحداث exception باستخدام واحداً من الثوابت المعرفة:

```
raise Exception.Create(SDiskFull);
```

Note

ملحوظة: ان النسخ الاولى من Delphi قد عرفت في الـ SysUtils unit الثوابت الموجودة في جدول (١٩-١) كمعرفي اعداد صحيحة من الـ resource، والتي يمكنك تمريرها الى الـ CreateRes constructor الـ Exception class. وهذه الثوابت قد تم تحديدها الآن كـ resource strings في الـ SysConst unit. ولكن مازالت الـ CreateRes متاحة لإنشاء الـ Exception objects التي تقوم بتحميل error message strings من جدول الـ Windows string resource-ولكن هذا الـ method يعتبر غير ملائم لأغلب تطبيقات Delphi.

جدول (١٩-١) : الـ SysConst Unit تعرف رسائل الخطأ المعيارية هذه، والتي يمكنك استخدامها لإنشاء الـ Exception Class الخاصة بك

Identifier	***String***
SinvalidInteger	'"%s" is not a valid integer value';
SinvalidFloat	'"%s" is not a valid floating point value';
SinvalidDate	'"%s" is not a valid date';
SinvalidTime	'"%s" is not a valid time';
SinvalidDateTime	'"%s" is not a valid date and time';
StimeEncodeError	'Invalid argument to time encode';
SDateEncodeError	'Invalid argument to date encode';
SOutOfMemory	'Out of memory';
SinOutError	'I/O error %d';
SfileNotFound	'File not found';
Identifier	String
SinvalidFilename	'Invalid filename';
StooManyOpenFile	Too many open files';
SAccessDenied	'File access denied';
SEndOfFile	'Read beyond end of file';
SDiskFull	'Disk full';
SinvalidInput	'Invalid numeric input';
SDivByZero	'Division by zero';

SRangeError	'Range check error';
SintOverflow	'Integer overflow';
SinvalidOp	'Invalid floating point operation';
SZeroDivide	'Floating point division by zero';
SOverflow	'Floating point overflow';
SUnderflow	'Floating point underflow';
SinvalidPointer	'Invalid pointer operation';
SinvalidCast	'Invalid class typecast';
SAccessViolation	'Access violation at address %p. %s of address %p';
SStackOverflow	'Stack overflow';
SControlC	'Control-C hit';
SPrivilege	'Privileged instruction';
SOperationAborted	'Operation aborted';
SException	'Exception %s in module %s at %p.'#$0A'%s%s';
SExceptTitle	'Application Error';
SinvalidFormat	Format "%s" invalid or incompatible with argument';
SArgumentMissing	'No argument for format "%s"';
SinvalidVarCast	'Invalid variant type conversion';
SinvalidVarOp	'Invalid variant operation';
Identifier	String
SDispatchError	'Variant method calls not supported';
SReadAccess	'Read';
SWriteAccess	'Write';
SResultTooLong	'Format result longer than 4096 characters';
SFormatTooLong	'Format string too long';
SVarArrayCreate	'Error creating variant array';
SVarNotArray	'Variant is not an array';

SVarArrayBounds	'Variant array index out of bounds';
SExternalExceptinn	'External exception %x';
SAssertionFailed	'Assertion failed';
SIntfCastError	'Interface not supported';
SAssertError	'%s (%s, line %d)';
SAbstractError	'Abstract Error';
SModuleAccessViolation	Access violation at address %p in module "%s". %s of address %p';
SCannotReadPackageInfo	'Cannot access package information for package "%s"';
sErrorLoadingPackage	'Can''t load package %s.'#13#10'%s';
SInvalidPackageFile	'Invalid package file "%s"';
SInvalidPackageHandle	'Invalid package handle';
SDuplicatePackageUnit	'Cannot load package "%s." It contains unit "%s,"' + ';which is also contained in package "%s"';
SWin32Error	'Win32 Error. Code: %d.'#10'%s';
SUnkWin32Error	'A Win32 API function failed';
SNL	'Application is not licensed to use this feature';

والمتبقى من الـ Exception constructors يعتبر صور من الاهداف السابقة . فيما يلى نماذج منهم . يقوم الـ CreateResFmt بتحميل string-table resource مع امر string format %s (لاحظ انه يجب عليك تمرير مجموعة string الى هذا الـ constructor) . يضيف الـ CreateHelp help-context المعرف (واستخدامه امر يرجع الى الـ exception handler الخاص بك) . الـ CreateFmtHelp يجمع الـ string formating مع help-context المعرف . والـ CreateResHelp يجمع string-table resource مع help-context المعرف . ويضع الـ CreateResHelp كل هذه الاشياء مع الـ string resource والذى يحتوى على formating codes مثل الـ %s أو %d ومعامل help-context:

```
raise Exception.CreateResFmt(SInvalidInteger, [IntToStr(N)]);
raise Exception.CreateHelp('Whoops, sorry', 123);
raise Exception.CreateFmtHelp('Error N=%d', [N], 123);
raise Exception.CreateResHelp(SFileNotFound, 123);
raise Exception.CreateResFmtHelp(SInvalidInteger, [S], 123);
```

تقنيات Exceptional اخرى:

فى الفصول التالية سوف نشرح تقنيات exception متقدمة تكون مفيدة فى الظروف الخاصة.

Silent exceptions:

ان الـ Silent exceptions هو exceptions ليس له تأثير ظاهر على البرنامج. ولكنه مفيد فى إفشال العمليات المتداخلة بعمق. ان silent exception instance هى object من الـ EAbort class، والمشتقة من الـ Exception class. وان برنامج معالجة الـ exception الافتراضى لـ Delphi قد تم برمجته على الا يعرض أى رسائل خطأ عندما يتلقى الـ EAbort exception. بدلاً من ذلك، يقوم البرنامج المعالج الافتراضى بتدمير exception instance ويعيد التحكم الى البرنامج.

قم باستدعاء الـ Abort procedure لاحداث exception من الـ EAbort class. على سبيل المثال، ان الـ while loop التى تفحص الـ NormalCondition Boolean flag. يمكنها ايضاً ان تفحص flag آخر، وهو الـ SpecialConditionFlag، وتخرج من الـ loop باستدعاء الـAbort :

```
while NormalCondition do
begin
  if SpecialConditionFlag = False then
    Abort;
end;
```

ان استدعاء الـ Abort يعتبر مساوياً لإحداث الـ EAbort exception class، والذى قد تقوم به بالطريقة الصعبةلتمرير رسالة مع الـ object (ولكن لان الـ exception يكون صامتاً، فلا يعرض البرنامج ابداً هذا الـ string):

```
raise EAbort.Create('Operation aborted');
```

ان اهمية هذه التقنية هى انها لا تنتج أى تأثير يظهر على الشاشة . تنتهى الـ loop بهدوء بأحداث exception من الـ EAbort class ، والذى يتلقاه برنامج معالجة exception الافتراضية ويعيد التحكم الى البرنامج . ان تمرير string بهذه الطريقة فى silent exception يمكن ان يكون مفيداً فى عملية (debugging) على سبيل المثال ، اذا كان برنامجك يحتوى على exception handler ، فإن هذا البرنامج المعالج قد يستجيب للـ EAbort كما يفعل مع أى exception اخرى . وبالتبادل اذا كنت لا تريده ان يتعرف على الـ silent exception (لذا فهى تبقى صامتة) ، استخدم الـ try-except block التالى ، والذى يعالج الـ exception ولكن يستدعى الـ ShowMessage فقط اذا لم يكن الـ exception من الـ EAbort class :

```
try
  { Statements that may call Abort }
except
  on E: Exception do
      if not (E is EAbort) then
          ShowMessage(E.Message);
      raise;
end;
```

استبدال برنامج معالجة الـ exception الافتراضى :

للحصول على عمل متقدم بحق ، يمكنك ان تبدأ فى البحث عن برنامج معالجة exception افتراضى مخصص يعمل على مستوى الـ Application وهذه التقنية توفر وصولاً الى كل الـ exception التى لم يتم معالجتها ، وهو ما قد يكون مفيداً فى إزالة اخطاء امكانات الـ exception handler الخاصة بالتطبيق وكذلك استبدال dialog box برنامج المعالجة الافتراضى مع المزيد من الوصف والايضاح .

حتى اذا لم تحتاج الى كتابة exception handler افتراضى لتطبيقك ، فقد تريد ان تجرب التقنية الموجودة فى هذا الفصل . وان تكون قادراً على الامساك بالـ exception التى لم يتم معالجتها على مستوى التطبيق هو أمر غاية فى القيمة وذلك

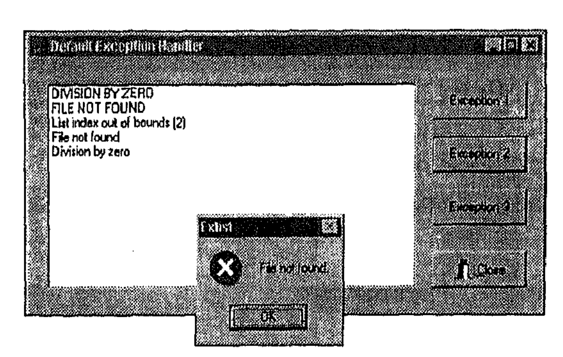
لإزالة أخطاء برنامجك. ان التطبيق بهذا الفصل يستخدم هذا الـ method للحفاظ على قائمة بكل الـ exception التى لم يتم معالجتها اثناء البرنامج. وهذه القائمة ذات قيمة عالية فى تتبع مصدر المشكلات الصعبة.

ان الـ TApplication يوفر procedure، وهو الـHandleException، والذى يقوم بمعالجة أى exceptions لم يتم معالجتها والتى تم حدوثها اثناء تنفيذ البرنامج. لا يجب عليك ان تحاول تجاهل هذا الـ method-فإن هذا سوف بتجنب تماماً برامج معالجة الـexceptions الافتراضية لـ Delphi، ومن المؤكد ان يؤدى الى مشكلة. بدلاً من ذلك، لزيادة برنامج المعالجة الافتراضى، يمكنك توفير procedure لـ event Application OnException ان الـ Handle Exception method يستدعى OnException procedure، اذا تم تعيين واحداً، بدلاً من عرض رسالة خطأ افتراضية.

On the CD-ROM

على القرص المدمج: يوضح شكل (١٩-١١) تنفيذ تطبيق الـ ExList. تمر كل الـ exception التى لم يتم معالجتها من خلال الـ handler الجديد، والذى يضيف فى هذا البرنامج رسائل الخطأ الى الـ ListBox. هذا يعطى تتبعاً تاماً للـ exception التى لم يتم معالجتها فى التطبيق يوجد البرنامج على القرص المدمج فى دليل الـ Source\ExList. قم بتشغيل هذا البرنامج واضغط ازرار الـ Exception الثلاثة لتوليد ثلاثة انواع من الـ exception. بعد ان تغلق dialog box الافتراضى الناتج، يضيف البرنامج رسائل خطأ.

أن البرنامج مضيف رسائل الـ exception error الى الـ ListBoxform. ان ضغط الـ Exception1 يحدث exception القسمة على صفر (أو divide-by-zero). ان ضغط الـ Exception2 يحدث exception عدم العثور على الملف (أوfile-not-found) ان ضغط الـ Exception3 يحول كل الـ strings المذكور الى حروف كبيرة- ولكن البرمجة تحتوى على خطأ يحدث index-out-of-bounds exception. توضح القائمة (١٩-١٥) الـ source code. للبرنامج.



شكل (١٩-١١): تطبيق الـ ExList يوضح كيفية كتابة برنامج exception handler افتراضى

القائمة (١٩-١٥): ExList\Main.pas

```
unit Main;

interface

uses
  Windows, SysUtils, Messages, Classes, Graphics,
    Controls,
     Forms, Dialogs, Buttons, StdCtrls;

type
  TMainForm = class(TForm)
      ListBox1: TListBox;
        Button1: TButton;
        Button2: TButton;
        Button3: TButton;
        BitBtn1: TBitBtn;
        procedure FormCreate(Sender: TObject);
        procedure Button1Click(Sender: TObject);
        procedure Button2Click(Sender: TObject);
        procedure Button3Click(Sender: TObject);
      private
      { Private declarations }
      public
```

```
{ 1. Declare the OnException event handler }
  procedure NewOnException(Sender: TObject;
        E: Exception);
          { Public declarations }
      end;

var
 MainForm: TMainForm;

implementation

{$R *.DFM}

{ 2. Implement the OnException event handler }
procedure TMainForm.NewOnException(Sender: TObject;
 E: Exception);
begin
 Application.ShowException(E);
    ListBox1.Items.Add(E.Message);
end;

{ 3. Assign the event handler to Application.OnException }
procedure TMainForm.FormCreate(Sender: TObject);
begin
 Application.OnException := NewOnException;
end;

{ Raise a divide-by-zero exception }
procedure TMainForm.Button1Click(Sender: TObject);
var
 I, J, K: Integer;
begin
 I := 0;
    J := 10;
    try
     K := J div I;  { Divide by zero! }
```

```
        { The following statement doesn't execute, but it is
           needed so the optimizer in Object Pascal, which
          notices
              that K isn't used in this procedure, doesn't strip
              out the preceding statement. Smart compiler. }
            ShowMessage('K=' + IntToStr(K));
      except
    raise;
      end;
end;

{ Raise a file-not-found exception }
procedure TMainForm.Button2Click(Sender: TObject);
var
 T: TextFile;
begin
 AssignFile(T, 'XXXX.$$$');
    Reset(T); { Open a non-existent file! }
    try
      { You would normally use T here }
      finally
     CloseFile(T); { For safety's sake }
      end;
end;

{ Raise an index-out-of-bounds exception }
procedure TMainForm.Button3Click(Sender: TObject);
var
 I: Integer;
begin
 with ListBox1.Items do
    for I := 0 to Count do { Should be Count - 1! }
   Strings[I] := Uppercase(Strings[I]);
end;

end.
```

ان الـ Application event handle يتطلب عـمـلاً أكثر من إنشـاء واحـداً، فلنقل مثلاً الـ Button OnClick . وهذا لان الـ Application object لا يمكن ان يتـوصل إليــه الـ Object Inspector الخـاص بـ Delphi's . لذلك فـإن إنشـاء OnException يتطلب ثلاث خطوات، والمرقمة فى التعليقات الموجودة بالقائمة (وهذه التعليقات مكتوبة بخط سميك ليسهل العثور عليها) وهذه الخطوات هى :

١- قم بتعريف OnException فى الـ form class . يجب ان يكون له الـ parameters الموضحة فى القائمة، بالرغم من ان تحديد اسم الـprocedure يرجع إليك .

٢- قم بتنفيذ OnException . تذكر ان تسبق اسمه باسم الـ form class نقطة .

٣- قم بتعيين الـ event handler للـ Application.OnException . ان الـ HandleException method الخاص بالـApplication object يتـأكـد مما اذا كان البرنامج قد قام بتعيين procedure للـ OnException . اذا كان كذلك، يقوم الـ HandleException باستدعاء الـ procedure لأى exceptions يتلقاه لم يتم معالجته .

ملحـوظة: ان الـ custom OnException event handler يتلقى الـ exceptions المشـتـقـة فـقط من الـ Exception class فـيـمـا عـدا الـ EAbort exceptions التى تبقى صامتة .

يمكنك اداء أى اعـمـال تريدها فى الـ OnException event handler . فى البـرنامج [انظر الخطوة رقم (٢)] تقـوم العـبـارة الاولى باسـتـدعـاء الـ ShowException method الخـاص بالـ Application . وهـذا مـا يقـوم به البرنامج الافـتـراضى بالفعل . بالإضـافـة الى ذلك، يسـتـخـدم البـرنـامج الـ Add method لقائمة الـ Items string الخاصة بالـ ListBox1 لحفظ نسـخـة من كل رسـالة خطأ لاحظ ان الـ event handler يتلقى الـexception object على انه parameter E . (هذا الـ object يتم تدميره فى أى مكان آخر، فـلا يجب عليك تحريره) .

يمكنك ان تجعل الـ event handler الافتراضى صامتاً بحذف الاستدعاء للـ Application ShowException. ولكن لا تفعل هذا دون تمييز ، فقد يخفى عيب خطير فى الـ code ان OnException الخاص بك يجب ان يعرض نوع من الرسائل عند تلقى exception لم يتم معالجتها .

افكار للمستخدم الخبير

● يجب أن تفكر فى الـ exception ، كأنما معدة لاعادة انواع متعددة من الـ object من الـ procedures والـ functions . من الناحية الفنية ، يمكنك ان تحدث exception لتمرر objects وتستخدم الـ try-except blocks للامساك بهذه الـ objects . إننى لا ارشح لك ان تستخدم الـexception بهذه الطريقة . ولكن من المفيد ان تدرك ان استخدامها للتحكم فى الخطأ هو امر ضرورى . ان معنى الـ exceptions objects ، خاصة تلك التابعة لـ classes خاصة بك ، يرجع إليك لتحدد تعريفه .

● ان الـ HandleException method الافتراضى فى الـ TApplication يرسل رسالة الـ wm_CancelMode الخاصة بالـ Windows لإطلاق الفأرة اذا كانت مقيدة فى وقت حدوث الـ exceptions . وهذا ايضاً يؤدى الى إغلاق list boxes ، والـ combo boxes، و drop-down lists ، والقوائم .

● عند إنشاء أى اشتقاق لـ class من الـ exception classes ، فضع بحذر عبارات on-do والتى تشير الى الـ class لمعالجة exception objects لأغلب الـ classes المشتقة من الـ Exception أولاً . فكر فى الـ code التالية ، والتى تشبه الامثلة الاخرى فى هذا الباب (ان عبارة الـ ShowMessage تستخدم الـ K لذا فإن الـ optimizer الخاص بالـ Object Pascal لا يحذف عبارة القسمة السابقة) :

```
try
  K := J div I;
    ShowMessage('K=' + IntToStr(K));
except
  on E: EDivByZero do  // This must come first!
```

```
        ShowMessage('Divide error');
         on E: EIntError do   // Trap all other EIntErrors here
        ShowMessage('Other integer math error');
    end;
```

● تستخدم هذه الـ code لاثنين من عبارات الـ on-do (يمكن ان يكون لديك أى عـــدد فى الـ except block). تمسك العـــبـــارة الاولى الـ EDivByZero exceptions، ربما لتفعل شيئاً خاصاً مع هذه الانواع من الاخطاء. وتمسك العبارة الثانية بكل الـexception المشتقة من الـ EIntError class، والتى تعتبر ايضاً الـ class الاسـاسـيـة للـ EDivByZero . وسيكون من الخطأ ان تعكس عـبارات الـ on-do لأنه اذا كانت الـ exception للـ class الام قد تم الامساك بها اولاً، فإنها سوف تشمل الـ EDivByZero exceptions (والمشتقة من الـ Exception) ولن يتم تنفيذ code خطأ الـ Divide الخاصة.

● ان تطبيق Delphi له إثنين من برامج معالجة الـ exception الافتراضية. يتم توفيـر واحداً منها بواسطة الـVCL ويحـمى كل procedure تمر من خـلاله الرسالة ويمسك هذا البرنامج المعالج بكل الـ exception التى لم يتم معالجتها، ويعرض رسالة خطأ، ويستمر فى تنفيذ البرنامج. اما الـ exception handler الافتراضى الآخر فموجود فى الـ RTL (مكتبة وقت التشغيل)، وتوفره الـ SysUtils unit. ويأتى هذا البرنامج المعالج فى مكانة اقل من البرنامج المعالج الافتراضى فى الـVCL، لذا يندران نراه عاملاً. عندما يمسك هذا البرنامج المعالج بـ exception، فإنه يعرض رسالة مفصلة (مع معلومات عنوان الـ exception لإزالة الخطأ)، ثم ينهى التطبيق.

المشروعات التى يمكنك تجربتها

(١٩-١): اعد كتابة القائمة (١٩-١٣) لاستدعاء function تحول خاصية الـ Edit1 Text الى قيمة عدد صحيح. هذا يوضح كيف يمكنك، مع الـ exceptions، ان تقوم بتعديلات كبيرة لبرنامج وتظل واثقاً من منطق معالجة الاخطاء الخاص بك.

(١٩-٢) : استبدال برنامج معالجة الـ exceptions الافتراضى للـ Application بالنسخة الخاصة بك التى تعرض dialog box اكثر معاونة. على سبيل المثال، قد يحتوى الـ dialog الخاص بك على زر Help يمكن للمستخدمين ان يضغطوه للحصول على معلومات حول انواع معينة الـ exceptions .

(١٩-٣) : قم بتحويل تطبيق الـ ExList الى تركيبة يمكنك دمجها فى أى تطبيق لإزالة اخطاء الـ exceptions التى لم يتم معالجتها . اضف امراً الى الـ module الخاصة بك لحفظ قائمة رسائل الـexceptions فى ملف نص .

ملخص:

- ان المطورين ذوى الخبرة يقومون بدمج code معالجة الخطأ فى برامجهم اثناء كتابتها . ان آليات الـ exception-handling الخاصة بـDelphi تجعل هذا الامر سهلاً وبسيطاً .
- يمكن ان تحدث الـ expressions من عدة مصادر . تستطيع الـ Components ، والتعبيرات الرياضية ، وعمليات الادخال/ والاخراج من الملف ، وقصور الذاكرة واحداث اخرى تحدث expressions . يمكنك ايضاً ان تحدث expressions لتخبر تطبيقك عن مشكلة ما .
- ان Protected-statement blocks تستخدم الكلمات الاساسية try-except لمعالجة exceptions قد تحدثها العبارات الموجودة فى جزئية الـ try .
- تستخدم الـ Protected-resource block الكلمات الاساسية try-finally لتنفيذ code إجبارية مثل تحرير ذاكرة مخصصة اذا حدث exceptions لعبارة فى الـ try block .
- ان الـ exception هو object من class ، تكون مشتقة غالباً من الـ class Exception ، تصف طبيعة الـ exception عند حدوث exception يؤدى الى إنشاء هذا الـ objectومعالجة الـ exception تؤدى الى تدميره . لا يجب ان تحرر exception instance ما بوضوح .

- استخدام عبارات الـ on-do فى الـ except block لمعالجة انواع معينة من الـ exception عند استخدام هذه التقنية، تذكر ان أى exception لم يتم معالجتها من انواع غير مذكورة فى عبارات الـ on-do تؤدى الى إنهاء الـ procedure أو الـ function الحالية بعد الـ except block. استخدم الـ try-finally blocks لمسح أى resource مخصصة فى مثل هذه الحالات.

- لاحداث exception استدع raise مع exception instance تم إنشاؤها حديثاً. لإعادة حدوث exception بعد معالجته فى الـ except block، استدع raise بلا arguments.

- يمكنك انشاء exception classes خاصه بك، إما من لاشئ، أو باشتقاقها من الـ Exception. فى اغلب الحالات، يجب ان تشتق exception classes الخاصة بك من الـ exception classes لان برامج معالجة الـ exception من هذه الانواع فقط.

- توفر الـ Exception class عدداً من الـ Create constructors لانشاء رسائل الخطأ من strings وقيم string-table resources.

- قم باستدعاء الـ Abort لأحداث exception صامت من الـ EAbort class. ان برامج معالجة الـ exception الافتراضية لا تعرض dialog رسالة خطأ للـ EAbort exceptions.

- قم بزيادة برنامج معالجة الـ exception الافتراضى وذلك بتعريف وتنفيذ وتعيين OnException فى الـ class المشتقة من الـ TForm والخاصة ببرنامجك. انظر تطبيق الـ ExList على القرص المدمج لمعرفة التعليمات استخدم هذه التقنية لإزالة اخطاء الـ exception التى لم يتم معالجتها لبرنامجك، أو لاستبدال dialog رسالة الخطأ الافتراضى.

يقدم الباب القادم افكاراً وتقنيات اساسية للقراء المهتمين بكتابة components و ActiveX controls.

*****



الباب العشرون
إنشاء custom Components

محتويات هذا الباب:

- Components
- معلومات حول الـ Components
- معلومات حول الـ packages
- تطوير الـ Components
- فهم الـ component design
- إنشاء الـ ActiveX controls

عندما تكتسب مزيداً من مهارات Delphi، فقد تبدأ تتسأل كيف تعمل الـcomponents فى الحقيقة. أو، تكون لديك فكرة جيدة عن محاور نشط يصنع component رائع ان اغلب مطورى Delphi لا يحتاجون الى بناء components، ولكن اذا كنت مهتم بهذا الموضوع المتقدم، فإن هذا الباب يساعدك على البدء فى تصميم component.

فى الفصول التالية، سوف تتعرف على أجزاء الـ component وكيف تتفاعل مع بيئة تطوير Delphi حيث يمكنك إنشاء، إختبار وتثبيت components جديدة فى لوحة Delphi components، ويمكنك إلقاء نظرة على العديد من تقنيات إنشاء الـ components الاساسية. يمكنك ايضاً تحويل احد الـ components فى هذا الباب الى .ActiveX control. بعد قراءة هذا الباب، سوف تكون مستعداً لان تذهب الى ماوراء الاساسيات فى تصميم الـ component وهو موضوع يمكن ان يملأ كتاباً بهذا الحجم.

معلومات حول الـ components:

ان الـ components هو Object Pascal class التى تلتزم ببعض القواعد واللوائح. يمكنك بناء components من component موجوداً بالفعل، ويمكنك إنشاء component من لاشئ. ولكن فى الغالب تقوم بتأسيس الـ component على تلك الـ component التى تأتى مع Delphi. على سبيل المثال، اذا اردت ان تضيف بعض التحسينات الى زر، يمكنك ان تؤسس component جديداً على الـ TButton class التابعة لـ Delphi وتضيف أى اشياء جديدة تريدها.

فيمايلى بعض الاسباب التى قد تجعلك تريد إنشاء components. هذه ليست قواعد جامدة- إنها مجرد اقتراحات قد تساعدك فى ان تقرر ما اذا كنت تريد إنشاء components.

● إنك تحتاج الى تعديل components موجود بالفعل لغرض خاص-على سبيل المثال، تريد Image components يستطيع ان يقوم بتحميل bitmap من الـ hardware التى تطورها شركتك. بتأسيس الـ component على الـ Image تكون كل خصائص و methods و events هذا الـ components متاحة فى تصميمك. يمكنك ان تشتق components جديدة من الـ TButton، TListBox، أو أى component class اخرى. هذا يؤدى ايضاً الى استخدام جيد للميراث المختص بالـ -object. فإن الـ class الجديدة ترث، إمكانيات الـ class الأساسى بدلاً من نسخ أو تعديل code موجودة بالفعل. ان الوراثة تساعدك على تنظيم مكتبات الـ components بشكل افضل، وكذلك تسهل الحفظ

● إنك تريد انشاء واجهة تطبيق Delphi component للـ control المخصص للـ Windows. عادة يمكنك ان تشتق هذا النوع من الـ TWinControl، والتى توفر الخصائص والـ methods والـ events الاساسية المشتركة فى كل الـWindows controls.

● إنك تريد ان تنشئ component، جديد تماماً من لاشئ هذا قد يحتاج الى بعض الجهد، ولكن قد توفر على نفسك كثير من الجهد والوقت بتأسيس الـ components على الـ TCustomControl، والتى توفر GDI Canvas مع الخصائص والـ methods والـ events العامة فى كل الـ components.

● ان يكون لتطبيقك متطلبات جرافيكية خاصة ، وتريد ان تنشئ components يوفر هذه الامكانيات . ان اغلب components الجرافيك تستند على الـ TGraphicControl class . وهناك method فى هذا الباب تستخدم هذا الـ method لإنشاء BarChart الذى يعرض BarChart من مجموعة من نقاط البيانات التى تريدها string list .

● إنك تريد ا ان تنشئ nonvisual component واحداً لا يظهر على الـ component palette . لقد قابلت بعض حالات من الـ nonvisual components - الـ TClipboard class مثلاً . إنها نادرة نسبياً ، ولكنها تستفيد من تقنيات إنشاء الـ components والتى تساهم فى قيمة البرنامج وسهولة الحفظ .

مؤلفى الـ components ومستخدميها:

عبر هذا الكتاب حتى الآن ، كنت انت مستخدماً للـ component . وحين تنتهى من هذا الباب ، ستكون مؤلفاً للـ component . ان مستخدمى الـ component ينشئون objects متعددة من الـ components فى وقت التصميم بإدخالها فى نافذة from ، وفى وقت التشغيل باستدعاء الـ Create methods. الخاصة بها يقوم مؤلفى الـ components بإنشاء components classes جديدة والتى يستطيع المستخدمون ان ينشئوا منها واحداً أو أكثر من الـ objects فى وقت التصميم وفى وقت التشغيل .

وهناك اختلافاً آخر بين مستخدمى الـ component ومؤلفى الـ component يختص بالوصول الى عناصر الـ component . كمستخدم component ، فإن لك حقوقاً مقيدة فى الوصول الى لب الـ component . على سبيل المثال ، يمكنك قراءة وكتابة الخصائص المنشورة فقط ، ويمكنك إنشاء events فقط لتلك الاعمال التى يتحها الـcomponent فى نافذة الـ Object Inspector .

كمؤلف للـ component ، فليس لديك هذه القيود . إنك تقرر أى الـ component تنشرها وكيف تنشئ الاعمال التى يحتاجها الـ component . ان كل عناصر الـ component متاحة لبرنامجك (فيما عدا الاعضاء الخاصة فى الـ classes الاساسية ، والتى كما تعرف ، يمكن التوصل إليها فقط فى هذه الـ classes) ، كما يمكنك الوصول الى العناصر المحمية فى الـ component الموجودة بالفعل . وفى

مقابل هذه الحرية، فإنك تقبل مسئولية كبيرة فى كتابةcode قوية . ان الاخطاء على مستوى الـ component يكون لها عواقب وخيمة أكثر منها فى اخطاء التطبيق .

ان إنشاء components خاصة بك :

- يوفر وصولاً الى الاجزاء المقيدة فى الـ components .
- يجعل من الممكن اضافة وتعديل خصائص وethods وevents لـ components موجودة بالفعل .
- يستلزم معرفة عملية جيدة بالـObject Inspector ، والبرمجة المختصة بالـ object ، ومعالجة الـ exception .

إجعل هدفك هو ان يكون الـ component أمنه الاستخدام حتى من قبل المبتدئين . اذا تطلب component ما عنصراً معيناً، فيجب ان يوفر قيمة افتراضية فى حالة اذا لم يقدم المستخدمون قيمة معينة . لا تفترض ان المستخدمين يعرفون كيف يقومون بنشر الـ component بشكل سليم . من الناحية النموذجية، يجب ان يكون المستخدمين قادرين على ان يفعلوا أى شئ بالـ components دون احداث مشكلة خطيرة .

بعض المصطلحات النافعة:

فيما يلى تعريفات للمصطلحات التى تحتاج ان تعرفها لإنشاء components . ان القاموس يلتزم باستخدام هذه المصطلحات فى المعلومات الاخرى المنشورة عن Delphi، ولكننى حاولت ان اكون صارماً هنا حول المعانى . الدقيقة للكلمات المتعلقة ببعضها مثل object و class ، أو declaration وdefinition وهذه الفروق الدقيقة وغيرها تهم مؤلفى الـ component الى حد كبير عن مستخدميها :

- **declaration**: هو عبارة لاسم عنصر، نوع، والـ parameters الخاص به .
- **definition**: هو انشاء أو تنفيذ عنصر موضح .
- **class**: هى تعريف نوع جديد يضم بيانات وcode للعمل على هذه البيانات . تمثل الـclass نوع بيانات بسيط مثل نوع الـ Object Pascal Integer . تبدأ اسماء الـ Class عادة بـ T (TButton ، مثلاً) .

- **Instance**: التعريف فى الذاكرة لمتغير class قد يعرف البرنامج أى عدد يحتاجه من الـinstances لـ class ما . وتشبه الـ instances متغيراً لنوع بيانات بسيط مثل الـ Integer. يمكنك تعريف أى عدد يحتاجه برنامجك من متغيرات الـ Integer، ولكن هناك نوع بيانات Integer واحد فقط . بنفس هذا المعنى ، يمكنك تعريف أى عدد يحتاجه برنامجك من الـ class instances . (ولكن بعض الـ classes النادرة قد تفيدك بانشاء instances واحدة . على سبيل المثال ، يمكن ان يكون هناك TApplication instances واحدة فى أى تطبيق) .

- **Component**: هو class مشتقة بطريقة مباشرة أو غير مباشرة من الـ TComponent ذات مسارات وصول خاصة لبياناتها ، وخصائصها المنشورة ، ومميزاتها الأخرى التى تلتزم بمتطلبات Delphi للـ components. إنك تقوم بتعريف وتنفيذ الـ components فى الـ Object Pascal units .

- **Object**: تماماً مثل الـ instance . فى النسخ السابقة من الـ Borland Pascal، كان الـ object هو والآن الـ class فى Delphi، مما يؤدى الى كثير من عدم الفهم فى الكتب والأدلة التى تنشر فى داك الوقت . فى هذا الكتاب ، لقد استخدمت كلمة object لتعنى بشكل محدد instance من class أو نوع آخر . لذا فمن الصواب ان يطلق على متغير الـ Integer انه object .

- **Access specifie**: هو واحدة من الكلمات الاساسية-private، protected، public، published- التى تقسم تعريف الـ class الى قطاعات . ان عبارات البرنامج لها درجات متفاوتة من حقوق الوصول الى التعريف ، وهذا يعتمد على محددى الوصول التابعين لهم . ان التعريف الـ private تكون للاستخدام فقط فى الـ unit التى تعرف الـ class ؛ وتكون التعريفات الـ protected ايضاً متاحة للـ classes المشتقة ؛ اما التعريفات الـ public فمتاحة لكل المستخدمين ؛ والتعريفات الـ published هى خصائص تعرف قواعد الوصول الى بيانات class، وبالنسبة للـ components المثبتة فى لوحة الـ VCl ، تظهر فى نافذة الـObject Inspector .

- **Field**: هو متغير معرف فى class .

- **Method**: هو procedure أو function معرف فى class .

- **Method implementation**: هو هيكل الـ method، بما فى ذلك أى متغيرات أو عبارات تؤدى اعمال الـ method .

- **Property**: هو حقل published يحدد بشكل نموذجى methods لقراءة وكتابة قيمة الحقل' وكذلك، هو published event handler مثل الـ OnClick .

- **Event**: هو pointer لـ procedure متوفر بواسطة المستخدم لـ objects معينة . قارن هذا بالـ method الذى يؤدى العمليات المتطابقة لكل الـ object للـ method class . ان خاصية الـ event تمكن الـ object ان يرسل اعمال الى object آخر، والذى يمكن ان يؤدى عمليات متنوعة لنفس نوع الـ event . على سبيل المثال، يمكن ان يستجيب اثنين من الـ TButton يعمل بنفس الطريقة مع كل الـ Button objects .

- **Unit**: هى Pascal module تم تخزينها وعمل compiled منفصلة عن الـ modules الاخرى. قد تحتوى units على component واحد أو أكثر. ولكن، يمكنك تثبيت units كاملة فقط، متضمنة كل الـ components، لوحة. الـ component . لتثبيت components منفردة، فيجب ان يكونوا فى units منفصلة .

- **Interface part**: هو التوضيحات التى يمكن التشارك فيها بشكل عام فى units ما؛ والتى تسبقها الكلمة الاساسية interface .

- **Implementation part**: هى التعريفات والتعليقات الخاصة التى لا يمكن التشارك فيها، وتكون مسبوقة بالكلمة الاساسية Implementation .

- **Initialization part**: هى عبارات تمر تنفيذها عندما تم تحميل الـ module فى الذاكرة؛ وتكون مسبوقة بالكلمة الاساسية initialization ومنتهية بالكلمة الاساسية end ونقطة .



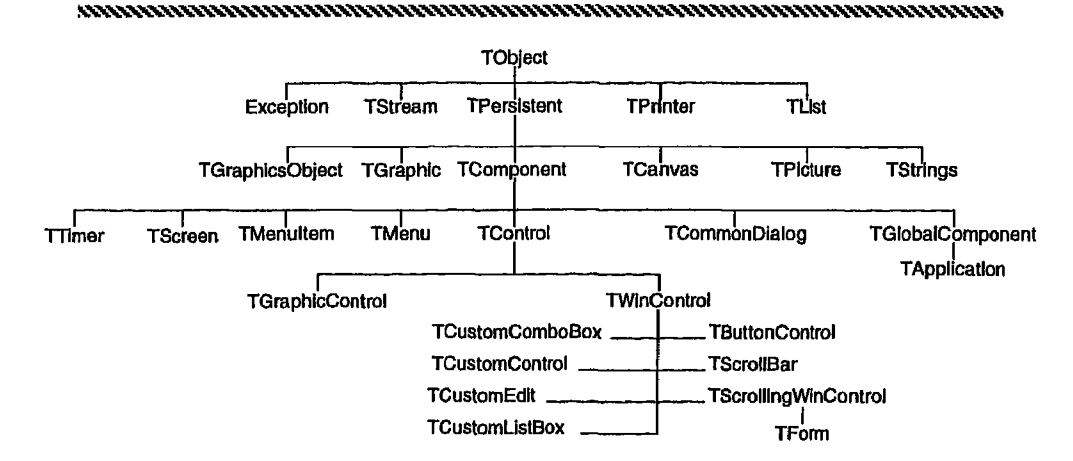
• **Visual component**: هو component له تمثيل بصرى فى وقت التشغيل. بعض الـ components مثل الـ TClipboard class تكون غير بصرية، ولكنها لا تزال components فى كل النواحى الاخرى. قد تظهر الـ components البصرية أو لا تظهر على الـ palette. على سبيل المثال، يعتبر الـ components TForm بصرياً، ولكنه غير متاح على الـ palette.

Note **ملحوظة**: من المفترض انك تدرك ماسأقوله الآن، ولكن الى الذين قفزوا الى هذا الباب، لقد اشرت فى هذا الكتاب الى الـ components الموجودة على الـ component palette باسماء نصها- مثل Button، ListBox وغيرها. ان classes هذه الـ components تسمى TButton و TListBox. عندما يكون الاسم مسبوقاً بحرف T، فإنه يشير الى تعريف لـ class؛ وبدون الـ T، فإنه يشير الى الـ components كما يتم استخدامه فى مصمم Delphi.

Component class hierarchy:

ان اغلب الـ components الجديدة تستند الى components' classes موجودة بالفعل. ويرث الـ component المشتق كل الخصائص والـ methods والـ events من الـ component السالف له. وللتمكن من إنشاء الـ components، فإنك تحتاج ان تتعرف على مكتبة Delphi component و methods التى توفرها لـ components يوضح الشكل (٢٠-١) الخاص بالـ classes hierarchy الذى يشكل الـ components السابقة. استخدم هذا الـ object لاستكشاف الخصائص والـ methods والـ events التى ترثها الـ class المشتقة. حاول الا تعيد اختراع امكانات يمكن ان ترثها من component class اخرى. وبالطبع ان الـ hierarchy الموضح هنا ليس كاملاً. إنك تجد شكل الشجرة الكامل مع Delphi.

لكى تصبح مؤلف components خبير، فإنك تحتاج الى الـ source files لمكتبة وقت التشغيل الخاصة بـ Delphi (ولكنك لا تحتاجهم لهذا الباب). ان الـ source files تشمل كل البرمجة الخاصة بمكتبة الـ component البصرى، ويمكنك تعلم تقنيات وحيل عديدة بتصفح الملفات.



شكل (٢٠-١): Delphi component class hierarchy

معلومات حول الـ Packages:

قبل تطوير component جديد، إنك تحتاج ان تتعرف على packages . والـ package يحتوى على compiled units ، والتى توفر الاعمال الداخلية لل components مثل ال Buttons والـ ListBoxes . وإنشاء package لهذه ال units يتم مهمتين اساسيتين :

- إنه يمكن تطبيقات متعددة من التشارك فى الـ VCL و component code اخرى، بدلاً من ان يطلب من كل تطبيق ان يكون له نسخة منفصلة من هذه الـcode .

- إنه يوفر وسيلة لتثبيت components فى Delphi palette . مع ال packages، ليس من الضرورى ان تعيد compile الـ VCL باكملها، كما كان الحال فى النسخ السابقة من Delphi، لتثبيت component فى الـ palette .

بالرغم من ان هاتين الميزتين تستخدما المستندات إلا انهما مختلفتين الى حد كبير ويستخدما package formats مختلفة لذا السبب يوجد نوعين من ال packages :

- **packages وقت التشغيل:** تستخدم لتوفير component code تستطيع تطبيقات Delphi المتعددة ان تتشارك فيها .



● **packages وقت التصميم:** تستخدم لتثبيت components في Delphi، وتوفر رابطة بمستندات وقت التشغيل المرتبطة بها .

يوضح هذا الفصل المفاهيم العامة لكيفية إنشاء واستخدام packages وقت التشغيل وpackages وقت التصميم . إنك بحاجة الى ان تفهم اساسيات الـ packages حتى تنشئ components جديدة بغرض تثبيتها على الـ Delphi palette.

Tip **فكرة:** لتشغيل packages وقت التشغيل، استخدم Project|Options، اختر الـ Packages page tab، وقت بتشغيل الـ Build مع packages check box وقت التشغيل يمكنك عندئذ بناء تطبيقاتك بحيث تتشارك في الـ code في units تكون موضوعة في مراجع من قبل الـ packages. وعند إبطال هذا الخيار، يتم نسخ الـ unit code مباشرة الى ملف code الـ exe.

ان إنشاء الـ packages. تعتبر أمراً بسيطاً. أولاً، قد بانشاء واختبار وازالة اخطاء الـ units التي تريد تثبيتها في packages. ان packages وقت التشغيل هي الاسهل من حيث الإنشاء لذا فمن الافضل ان تبدأ بها . قم بوضع units واحدة أو أكثر تريد تثبيتها في packages، ثم اتبع هذه الخطوات :

١- ابدأ تطبيقاً جديداً يمكن ان يكون هذا التطبيق هو البرنامج الاختياري للـ packages أو الـ units الخاصة بك، ولكن اذا كنت لا تحتاج واحداً، يمكنك إغلاق وتجاهل ملفات مشروع وunits التطبيق .

٢- اختر ..... File|New واضغط الـ New page tab. اضغط مرتين ايقونة الـ Package لتبدأ الـ Package Editor الخاص بـ Delphi.

٣- ان الـ New Package dialog يسأل عن اسم ملف والوصف . اضغط زر الـ Browse لفتح dialog ملف معياري، وانتقل الى الدليل حيث تريد تخزين ملفات المستند . قم بادخال اسم ملف في هذا الـ dialog بدون إمتداد اسم ملف، على سبيل المثال، TestPkg . إضغط Enter، ويدخل الـ Delphi مساراً واسم ملف زائد الامتداد بـ dpk.. في مربع اسم الملف . إدخل وصفاً اذا اردت - هذا يعتبر اختيارياً، ولكنه يظهر في اماكن متنوعة، فمن المستحسن ان تدخل واحداً .

٤- اضغط زر الـ OK للـ New Package dialog. هذا يؤدى الى إظهار نافذة الـ Package Editor الخاصة بـ Delphi. لاضافة units الى الـ Package، اضغط زر الـ Add، وتصفح ملف الـ pas source code. الخاص بالـ unit. لاحظ انك تضيف ملفات source code، وليس ملفات compiled unit، الى الـ Package، اضغط زر Ok واضغط Enter للعودة الى الـ Package Editor. اضغط الـ Add، مرة اخرى لأى عدد من الـ units تريد ان تضيفها الى الـ package.

٥- عندما تنتهى من اضافة الـ units، اضغط الـ Requires page tab الخاص بالمحرر. هذا يؤدى الى اظهار الـ package وهى تحتوى على الـ units التى تحتاجها و units. اخرى فى الـ package الجديد. وكحد أدنى، فإن الـ package تحتاج الى ان تشير الى VCL40 والذى يحتوى على الـ VCL units. لإضافة Package اخرى. اضغط زر الـ Add وقم بالدخول فيه أو تصفح ملف package آخر.

٦- عندما تنتهى من إضافة الـ units المحتواه والـ package المطلوبة، اضغط زر الـ Options. اضغط الـ Description page tab واختر واحداً من الـ check boxes اللذان يحملان labeled Usage options أو كلاهما. للحصول على هذا العرض، قم بتشغيل الـ Runtime package option فقط. لإنشاء وتثبيت components فى الـ Delphi palette، فإنك تقوم بتشغيل كلا الخيارين. قم بتحرير الخيارات الاخرى كما تدعو الحاجة للـ units وcomponent. اضغط OK لإغلاق الـ Package Options dialog.

ان هذه الخطوات تتم تصميم الـ package الجديد. (ان ادخال components فى الـ package والـ Delphi palette يتطلب بعض الخطوات الإضافية، والتى ساوضحها حالاً). إنك الآن مستعد لـ Compile واستخدام الـ package الجديد الخاص بك. أضغط زر الـ Compile لـ Compile الـ package، وكل الـ units هذا يؤدى الى إنشاء bpl package. فى الدليل المخصص.

لاستعراض الـ source code الخاصة للـ package اضغط يميناً داخل نافذة الـ Package Editor، واختر أمر الـ View Package Source. هذا يفتح الـ

package فى نافذة محرر Delphi code . وبتجاهل السلسلة الطويلة من خيارات الـ Compile، والتى يتم إدخالها بصورة تلقائية فى النص، يعتبر الـ package ببساطة عمل بسيط يبدو كهذا :

```
package TestPkg;
{$R *.RES}
{$ALIGN ON}
{$ASSERTIONS ON}
...
{$DESCRIPTION 'Test runtime package'}
{$DESIGNONLY}
{$IMPLICITBUILD ON}

requires
  vcl40;

contains
  Clrform;

end.
```

لقد استبدلت أغلب أوامر الـ compilation بزر بيضاوى- سوف ترى المجموعة الكاملة عندما تستعرض الـ source code للمستند الخاص بك .

تعتبر الـ Packages ملفات compiled DLLs بشكل خاص. ولتمييزها عن ملفات الـ dll. الأخرى، فإن امتداد اسم ملف الـ package هو bpl.. يحتوى ملف package source على ثلاثة عناصر أساسية :

- الكلمة الأساسية package واسم الملف متبوعاً بفصلة منقوطة .
- تعريف الـ required الذى يحتوى الـ packages الأخرى التى يطلبها هذا الـ packages .
- تعريف الـ contains الذى يحتوى الـ units التى يجب ضمنها فى هذا الـ package .

Note **ملحـوظـة:** بالرغم من التطبـيـق يكون مـرتبطاً بـ packages وقت التشغيل، إلا أنه يجب على modules البرنامج أن تضيف الـ units التى تحتاجها الى الـ uses . إن الـ Compiling مع packages وقت التشغيل يخبر التطبيق فقط أين يجد الـ component code . يجب أن تستمر الملفات الـ source للتطبيق فى إدخال الـ units التى تطلبها بذكر اسماء الـ units فى تعريف الـ uses.

إن الـ Components التى تريد تثبيتها على لوحة Delphi يجب أن تكون compiled باستخدام الـ packages . والخطوات المطلوبة لعمل هذا تشبه الخطوات الموضحة لـ packages وقت التشغيل، ولكنك عادة ماتختار مخرجات مستندات الـ packages Runtime والـ Design-time للـ components هذا يؤدى الى إنشاء packages : واحداً للاستخدام فى وقت التشغيل للتطبيقات الـ packaged components، وواحداً للاستخدام فى وقت التصميم لتشغيل الـ component على الـ VCL palette .

قد يكون من قبيل الفائض عن الحاجة أن يكون هناك نوعين مختلفين من الـ packages، ولكن فى الواقع، فـإن packages وقت التـصـمـيم تـشـيـر الى packages وقت التشغيل. إن packages وقت التصميم من مجرد غلاف- فإن الـ code الفـعليـة للـ unit تكون دائماً فى packages وقت التـشـغـيل هذا يعنى أنه بإمكانك توزيع packages وقت التـصـمـيم للمبرمجين الذين يحتاجون الى استخدام الـ components بإختيارها من VCL palette الخاصة بـ Delphi .

لفـتح ملف package لإعـادة الـ Compile، اخـتـر امـر الـ File|Open، واستعرض ملفات من نوع Delphi package source (*.dpk) .

Note **ملحوظة:** لاستخدام package وقت التصميم-والذى يجب ان تفعله لكل الـ components المثـبـتـة على الـ VCL-يجب ان يكون لديك package وقت التصميم المناسب وpackage وقت التشغيل المتعلق به .

يوضح الفـصل التالى كـيـفـيـة إنشـاء component جـديد، وادخـالـه فى package، وتـثـبـيـت الـ component VCL palette الخـاصـة بـ Delphi فى الـSamples .



تطوير الـComponent:

فى الفصول التالية، تقوم بإنشاء إثنين من الـ components التى توضح برمجة الـ component وعملية تثبيت component تام فى Delphi palette . كل هذه الخطوات يمكن الرجوع عنها.

اولاً، تقوم بانشاء component بسيط فقط لتوضيح بعض الخطوات المطلوبة. بعد ذلك، تقوم بإنشاء component أكثر تعقيداً يوضح المزيد عن الخصائص و الـ methods والـ events، زائد تقنيات الـregistration والـ debugging والـ installation .

الخطوات الأولى:

تتطلب الـ components تقنيات برمجة مختلفة عنها فى التطبيقات. يمكنك برمجة components من لاشئ باستخدام محرر Delphi code ، ولكن لان الـ components التى سيتم تثبيها على الـ VCL palette تتطلب التثبيت فى الـ Package ، فمن المستحسن ان تستخدم الـ Package Editor الخاص بـ Delphi لإنشاء الـ component unit يجب ان تستخدم هذا المحرر على ايه حال لـ compile وتثبيت الـ component فى الـpalette ، لذا فيمكنك ايضاً استخدامه لإنشاء الـ unit .

إتبع هذه الخطوات لتقسيم الـ Button component القياسى component جديد يحمل اسم TDingButton . وهذا الـ component يقوم بإنشاء زر والذى، عندما يضغط ، يدق جرس الحاسوب . هذا الاختيار يوضح إثنين من الجوانب الهامة فى بناء الـ component: وراثة إمكانات من component موجود بالفعل والذى تبنى عليه الـ component الجديد، واستخدام الـPackage لتثبيت component على الـ VCL palette .

On the CD-ROM **على القرص المدمج:** ان الدليل Source\Ding على القرص المدمج يحتوى على كل الملفات الخاصة بهذا الفصل. اذا لم تكن تريد إنشاء الـ component الخاص بك باتباع الخطوات المذكورة هنا، افتح ملف الـ Ding package، واستخدم الـ Package Editor لـ compile وتثبيت الـ

component فى Delphi. ولكن، لكى تفهم العملية جيداً، فإننى اقترح عليك ان تنشىء دليلاً جديداً وتتبع هذه الخطوات لبناء الـ TDing class. الخاصة بك.

١- قم بإنشاء دليل (لقد اعطيك الدليل الخاص بى اسم Ding لتحميل الـ component واختيار ملفات البرنامج. إبدأ تطبيقاً جديداً. إجعل اسم form MainForm. اختر ...File|Save all واحفظ الـ Unit1 على انها Main.pas، والمشروع على انه TestDing.dpr فى الدليل الخاص بك. يكون المشروع بمثابة برنامج اختيارى للـ component.

٢- اختر ...File|New واضغط الـ New page tab. اضغط مرتين ايقونة الـPackage.

٣- باستخدام الـ New Package dialog، اضغط Browse وتحقق من ان الدليل من الخطوة رقم (١) مازال جارياً. إدخل DingPkg فى حقل اسم الملف، واضغط Open. هذا يؤدى الى ادخال المسار واسم الملف DingPkg.dpk فى الـ New Package dialog. واختيارياً، قم بادخال وصف مثل "Button with a bell," أو، واضغط OK للاستمرار.

٤- ان الـ Package Editor الآن فى حالة تشغيل. اضغط زر الـ Add، عندما يظهر الـ Add dialog، اضغط الـ New Component page tab. هذا يؤدى لفتح dialog آخر يمكنك ان تحدد فيه صفات الـ component الخاص بك. إدخل أو اختر من قوائم اللائحة المعلومات التالية:

- (Ancestor type) : TButton .
- (Class name) : TDingButton .
- Palette page : Samples (الافتراضى).
- (Unit file name) : C:\...\Ding\DingButton.pas (يتم ادخاله بصورة تلقائية).
- (Search path) : (يجب ان يحتوى على مسارات افتراضية زائد المسار الخاص بـ Ding).

٥- اضغط OK لإنشاء غلاف الـ component . هذا يقفز بك الى الخلف الى Package Editor . اضغط الـ Requires وحدد أى package مطلوبة- لهذا . فى هذا المثال ، فإن vcl340 package الافتراضى هو كل ماتحتاجه . فى الـ components الخاصة بك ، قد تريد ان تضيف package اضافية الى هذه القائمة .

٦- اضغط زر الـOptions وقم بتشغيل كلاً من Design و Runtime check boxes . قد تغير فى الخيارات الأخرى ، ولكن لهذا العرض ، فإن القيم الافتراضية لابأس بها اضغط OK للعودة الى الـ Package Editor .

٧- اضغط زر الـ Compile لإجراء عملية الـ compile لـ package والـ component unit. الخاصة بك . هذا يؤكد ببساطة ان الـ format والـ unit. وفى الـ package الصحيحة لهما . عندما تكون مستعد لتثبيت الـ component التام ، اضغط Compile مرة أخرى ، ثم اضغط Install لادخال الـ component فى الـ palette . ولكن لا تفعل هذا الآن-اولاً ، نريد ان نبرمج ما يقوم به الـ component الجديد .

توضح القائمة (٢٠-١) ملف الـ DingButton.pas التام . انتقل الى نافذة محرر الـDelphi code ، واختر الـ DingButton page tab . استخدم القائمة كمرشد لك لملئ غلاف الـ component الذى انشأته فى الخطوات السابقة .

القائمة (٢٠-١) : Ding\DingButton.pas

```
unit DingButton;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics,
   Controls,
    Forms, Dialogs, StdCtrls;


type
  TDingButton = class(TButton)
    private
      { Private declarations }
       protected
```

```
    { Protected declarations }
     public
    procedure Click; override;
    { Public declarations }
     published
    { Published declarations }
     end;

procedure Register;

implementation

procedure Register;
begin
 RegisterComponents('Samples', [TDingButton]);
end;
procedure TDing.Click;
begin
 MessageBeep(0);
    inherited Click;
end;

end.
```

يضيف البرنامج التام سطور قليلة فقط الى قالب الـ component . لتوفير صوت جرس عندما يضغط المستخدمين الزر، فقد عرفت الـ Click procedure فى الـ TDing class مع امر (override) :

```
procedure Click; override;
```

ان الـ TButton class الذى يستند عليها الـ TDing تعرف الـ procedure Click مع العديد من الخصائص والـ methods الاخرى . ان تعريف override يخبر الـ compiler ان الـ component الجديد يحل محل الـ Click الموروث مع البرمجة الخاصة به . بالإضافة الى تعريف الـ method ، يجب عليك ايضاً تنفيذه بكتابة هيكل الـ procedure . فى هذا المثال ، فإن الـ Click method المستبدل ينفذ امرين :

```
MessageBeep(0);
inherited Click;
```

ان السطر الاول يصدر صوتاً باستدعاء الـ Windows API MessageBeep function. والسطر الثانى يستدعى الـ Click procedure المشتق لأداء اعمال الـ TButton، والتى تشمل تنفيذ OnClick اذا تم تعيين واحداً.

عند ابطال method، فإنك دائماً ما تستدعى الـ method المشتقه. هذه هى كيفية بنائك لإمكانات جديدة فى الـ component مع الاحتفاظ بما تقوم به الـ methods بالفعل. وليس مطلوباً منك ان تستدعى method مشتقاً، ولكن اذا لم تفعل، فقد يتخطى برنامجك شيئاً هاماً مدفوناً فى مكان ما فى الـclass الأم. للسلامة، استدع دائماً الـ method المشتق فى procedures و functions الاحلال الا اذا كان لديك سبب قوى فى الا تفعل.

لاحظ ان الـ DingButton.pas module تشمل الـ Register procedure والذى لا يعتبر عضواً فى الـ TDing class. يستدعى الـ Delphi هذا الـ procedure كجزء من عملية تثبيت الـ component palette الخاصة به اتبع خطوات تثبيت الـ componen الجديد (بعد ذلك، سوف اشرح كيف تحذف الـ procedure، لذا لا تتردد فى تجربة هذا).

١- ارجع الى نافذة الـ Package Editor. اذا لم تستطيع العثور عليها، اختر ...View|Window List واختر Package Editor لإظهار هذه النافذة.

٢- اضغط زر الـ Compile. هذا يؤدى الى Compile الـ package والـ unit المتعلقة به. تذكر، يجب ان تقوم بـ Compile الـ package ان Compile الـ unit. فقط منفصلة لا يؤدى الى إنشاء سليم للـpackage.

٣- اذا لم تتلقى ايه اخطاء فى الخطوة رقم (٢)، اضغط Install لتثبيت الـ component على الـ VCL palette. يجب ان تتلقى رسالة مثل "Package D:\...\DingPkg.bpl has been installed. The following new component(s) have been registered: TDingButton." اضغط زر الـ OK لاغلاق نافذة الرسالة هذه.

اضغط الـ Samples page tab على الـ VCL palette الخاصة بـ Delphi .
ان الزر الجديد موجود على page وجاهز للاستخدام لاختبار الـ component الجديد، يمكنك انشاء تطبيق جديد، أو استخدم التطبيق الحالى الذى انشأته فى بداية لعملية باكملها. لاستخدام هذا التطبيق، افتح الـ Project Manager اذا لزم الأمر واجعل الـ Main form مرئية. أضف DingButton فى الـ form . لاحظ ان Delphi يعطى هذا الـ object اسم DingButtonl. لاحظ ايضاً اذا اوقفت المؤشر على ايقونة الـ component ، فإن Delphi يعرض اسمه فى نافذة.

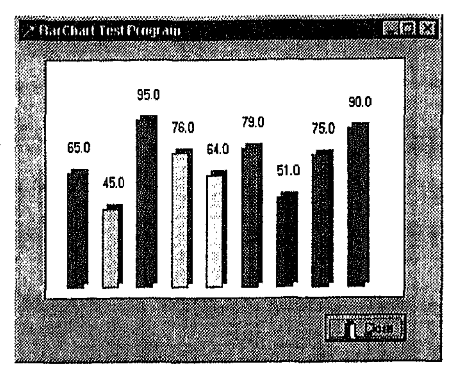
اضغط F9 لـ compile التطبيق. اضغط DingButton لتدق جرس الحاسوب الخاص بك. قد تريد ان تحفظ كل الملفات فى هذه المرحلة قبل الاستمرار.

لحذف component ، استخدم ...ComponentlConfigure Palette، اختر فئة على اليسار (Samples مثلاً)، واختر الـ component الذى تريد ازالته (وهو Ding فى هذه الحالة)، ثم اضغط Delete .

الخصائص، والـ methods، والـ events :

ان كتابة components يتضمن برمجة ثلاثة عناصر اساسية تكون موجودة فى كل class فى كل component وهى: الخصائص والـ methods ، والـ events. فى هذا الفصل والفصول التالية، فإنك تقوم بتطوير، واختبار، وتثبيت components اكثر توسعاً والتى تستخدم هذه العناصر الثلاثة. والنتيجة التامة، وهو الـ BarChart من مجموعة من نقاط البيانات تديرها قائمة الـ string list . يوضح شكل (٢٠-٢) مثال للـcomponent المستخدم.

على القرص المدمج: اذا لم تكن تريد إنشاء الـ component الخاص بك، انسخ دليل الـ Source\BarChart من على القرص المدمج، اختر ComponentlInstall Packages، اضغط Add، وتصفح دليل الـ BarChart على محرك القرص الصلب الخاص بك. اختر BarPkg.bpl، واضغط OK مرتين. ولكن، إننى اشجعك على اتباع التعليمات الموضحة هنا خطوة بخطوة، والتى توضح بطريقة افضل كيفية إنشاء وتثبيت الـ components الخاصة بك. (ولكن مازال بامكانك نسخ نص البرنامج من على القرص المدمج لتوفير الوقت).



شكل (٢٠-٢) : نموذج للـ BarChart الذى تقوم بتطويره واختباره وتثبيته فى Delphi

لإنشاء BarChart component اتبع الخطوات التالية :

١- قم بانشاء دليل BarChart جديد . ابدأ تطبيقاً جديداً، اجعل اسم الـ form MainForm، واحفظ جميع الملفات واحفظ الـ Unit1.pas على انها Main.pas ، واحفظ المشروع على انه Bartest.dpr . كما كان الحال مع زر الـ Ding، يكون الـ Bartest بمثابه برنامج اختبارى للـ BarChart .

٢- اختر ..... File|New، اضغط الـ New page tab، اضغط مرتين ايقونة الـ Package . وفى الـ New Package dialog الناتج، اضغط Browse ، انتقل الى دليل الـ BarChart ، وإدخل BarPkg فى حقل الـ File name . اضغط Open للعودة الى الـ New Package dialog. يجب ان ترى ...\BarChart\BarPkg.dpk فى الـ File name editor. ادخل BarChart Component فى حقل الـ Description واضغط OK .

٣- إنك ترى الآن الـ Package Editor. اضغط زر الـ Add ، ثم اختر الـ New Component page tab . حدد الحقول الموجودة فى صفحة الـ dialog هذه بمايلى :

* (Ancestor type) : TGraphicControl .
* (Class name) : TBarChart .
* (Palette page) : Samples (الافتراضى) .

* (Unit file name) : C:\...\BarChart\BarChart.pas (تأكـد مـرتين من هذا المدخل-فـقـد يكون مـحـدداً بدليل الـ Ding من الفـصـل السـابق. اذا كـان كذلك، قم بتغييره الى BarChart).

* (Search path) : (يجب ان يحتوى على مسارات افتراضية زائد المسار الخاص بالـBarChart؛ قم بحذف المدخل الخاص بالـ Ding اذا كان ضرورياً).

٤- اضــغط OK لإنشـاء غــلاف الـ component الجـديد، وارجع الى الـ Package Editor. اضغط الـ Requires page tab وحدد أى Package يطلبها هذا الـ Package. كـمـا كـان الحـال مع الـ Ding، ان كل المطلوب هو الـ vcl40 package الافـتـراضى. فى الـ component، قـد تحـتـاج ان تضـيـفـpackage اضافية الى هذه القائمة.

٥- اضــغـط زر الـ Options وقم بتـشـغـيل كـلاً مـن الـ Runtime check boxes و الـ Design. ان القيم الافتراضية للخيار الآخر لابأس بها. اضغط OK للعودة الى الـ Package Editor.

٦- اضـــغط زر الـ Compile لـCompile الـ package والـ component الخاصة به. هذا يضمن ان الـ package و الـ.unit فى الـ formats الصـحيـحة لهمـا. عندما تكون مستعداً لتثبيت الـ component اضغط Compile مرة أخيرة، ثم اضـــغط Install لإدخـال الـ component فى اللـوحـة. ولكن، لا تـفـعـل هذا الآن-كـمـا كـان الحـال مع زر الـDing، فنحن نـحـتـاج اولاً ان نبرمج مـا يقـوم به الـ component الجديد.

٧- انتـقل الى Delphi code editor، واخـتـر الـ BarChart page tab . انسخ البرمـجة من على القرص المدمج فى ملف BarChart.pas الموجـود فى دليل الـ Source\BarChart. ان باقى هذا البـاب يشرح عبارات هذه القـائمة-امـا الآن، اسـتـخـدم النص كله ليـحل مـحـل الـ BarChart.pas unit الفـارغـة فى الـcode editor.

ان الخطوات السابقة تتم البرمجة للـ BarChart component. يمكنك الآن compile الـ component وتـثــبــيت الـ package الخـــاص به على الـ VCL palette. بعـــد ذلك، يمكنك اســتــخــدام الـ component كـمــا تفـعـل مع الـ



components الاخرى . لاكمال الـ component المخصص واختباره ، إتبع هذه الخطوات :

١- ارجع الى نافذة الـ Package Editor . اذا لم تستطيع العثور عليها ، اختر ...View|Window List واختر Package Editor لإظهار هذه النافذة .

٢- اضغط زر الـ Compile . هذا يؤدى الى Compile الـ package والـ unit المتعلقة به. تذكر ، يجب ان تقوم بـ Compile الـ package ان Compile الـ unit فقط منفصلة لا يؤدى الى إنشاء سليم للـpackage .

٣- اضغط Install لتثبيت الـ component الجديد فى الـ VCL palette . يجب ان تتلقى رسالة مثل "Package C:\..\BarPkg.bpl has been installed. The following new component(s) have been registered: TBarChart." اضغط زر الـ OK لاغلاق نافذة الرسالة هذه .

٤- قم بتجربة component الجديد . اضغط الـ Main page tab فى نافذة الـ code editor ، واضغط F12 لاظهار form هذا البرنامج الاختبارى . اضغط الـ Samples VCL ، واختر الـ BarChart . اضغط فى الـ form لإضافة الـ BarChart .

٥- مع اختيار الـ BarChart1 ، اضغط الزر البيضاوى الواقع بعد خاصية الـ Data للـ object . هذا يؤدى الى اظهار محرر قائمة الـ string الخاص بـ Delphi . إدخل قيم floating-point من صفر الى مئة ، مدخل واحد لكل سطر (٩ , ٣٢ ، ١٢,٣)٠ , ٧٥ ، ٩٨,٧ ، الى آخره) .

٦- اضغط زر OK الخاص بمحرر string القائمة للعودة الى الـ form واستعرض الـ chart المعروض بواسطة الـ BarChart . يوضح شكل (٢-٢) الـ BarChart الذى تراه على الشاشة (لايجب عليك تشغيل أو حفظ البرنامج الاختبارى ، ولكن تستطيع اذا اردت) .

الـ TBarChart component class :

توضح القائمة (٢٠-٢) قطاع واجهة التطبيق للـ BarChart.pas unit ، بما فى ذلك تعريف الـ TBarChart class . كما ترى ، فإن الـ Barchart unit

تستخدم العديد من Delphi unit modules الاخرى، ويمكنك اضافة اسماءunit اخـرى الى تعـريف الـ uses . ولأنك لا تقـوم ببـرمــجــة form ، فـعند برمـجـة component ، لايستطيع Delphi ان يضيف بصورة تلقائية اسماء الـ units الى الـ uses كما يفعل مع التطبيق. اذا قمت باستدعاءprocedure أو function ، أو اذا حاولت ان تنشئ object لـ class ، وتلقيت خطأ Unknown identifier ، انظر نوع identifier فى online help الخاصـة بـ Delphi واضف unit إلى uses فى اعلى الـunit module . يمكنك ايضـاً اضـافـة تعـريف uses خـاص للـ unit module .

على القرص المدمج: على خلاف اغلب قوائم المشروعات الموجودة فى هذا الكتاب، قائمة الـ BarChart مذكورة فى هذا الباب فى اجزاء. سـوف اوضح كل جـزء عندمـا تمر به. وبالقطع، توجـد قـائمـة الـ BarChart.pas باكملها على القرص المدمج فى دليل الـ Source\BarChart .

القائمة (٢٠-٢): Barchart\BarChart.pas interface

```
unit Barchart;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics,
    Controls,
      Forms, Dialogs;

type
  TBarChart = class(TGraphicControl)
    private
      FPen: TPen;
        FBrush: TBrush;
        FData: TStrings;
        FLabels: Boolean;
        XBase, YBase: Integer;
```

```
        XIncrement, YIncrement: Integer;
          procedure SetPen(Value: TPen);
          procedure SetBrush(Value: TBrush);
          procedure SetData(Value: TStrings);
          procedure SetLabels(Value: Boolean);
          function YData(N: Integer): Integer;
    protected
        procedure Paint; override;
         public
        constructor Create(AOwner: TComponent); override;
          destructor Destroy; override;
         published
        procedure StyleChanged(Sender: TObject);
          property Pen: TPen read FPen write SetPen;
          property Brush: TBrush read FBrush write
        SetBrush;
          property Data: TStrings read FData write SetData;
          property Labels: Boolean read FLabels write
        SetLabels;
          property DragCursor;
          property DragMode;
          property Enabled;
          property ParentShowHint;
          property ShowHint;
          property Visible;
          property OnDragDrop;
          property OnDragOver;
          property OnEndDrag;
          property OnMouseDown;
          property OnMouseMove;
          property OnMouseUp;
         end;

  procedure Register;
```

ان الـ TBarChart (الـ class المشـتـقـة) ترث خـصـائص وevents و methods من الـTGraphicControl (الـ class الأم) كـمـا ترى من الشكل (٢٠-١)، ان الـ class الام مشتقة من اربع classes اخرى : الـTControl ، الـ TComponen، الـTPersistent ، والـTObject . ولـذلـك فـإن الـ TBarChart ترث كل خصائص وmethods و events تلك الـclasses . ولان العديد من الـ components تكون مشتقة ايضاً من نفس هذه الـclasses . ، فإن كل الـ methods والـ events المشتقه تكون معلقة فى مكان ما فى تطبيقك. فإن وراثة هذه الاشياء تعنى ان الـclass الجديدة يمكن ان تتوصل إليها.

تبدأ الـ TBarChart class بعدة تعريفات خاصة. فإن كل الحقول الـ methods التى تعقب الكلمة الاساسية private هى مخصصة للاستخدام من جانب الـ unit التى تعرف الـ class. ولا تشير أى unit اخرى فى أى مكان- أو component أو تطبيق-الى أى من التعريفات الخاصة للـclass. ويعد هذا مثالاً على إخفاء البيانات، وهو الأمر الذى يدرك المبرمجين الخبراء انه واحداً من اسس التطوير الناجح للبرمجيات. وتحدث (Bugs) نتيجة استخدام البيانات بلا تمييز، وإخفاء عناصر هامة (مثل قائمة الـ FData string الخاصة بالـ BarChart) يضمن أنه اذا ظهر أى خطأ بهذا الحقل، فيجب ان تكون المشكلة فى هذه الـ module، لان هذه الـmodule وحدها تستطيع ان تشير الى الـ FData. ان اخفاء البيانات ايضاً يجعل المطور حراً فى ان يقوم بتعديل الـ class code دون قلق أن تؤثر هذه التغييرات على modules أخرى.

ولكن، تحتاج الـ TBarChart class أن تجعل قائمة الـ FData string متاحة بحيث يستطيع مستخدمى الـ component أن يوفروا بيانات جديدة لـ bar chart objects. وتقوم الـ class بهذا العمل من خلال نشر الحقل، والذى يحدد methods الوصول التى تحكم كيفية قراءة المستخدمين للبيانات الهامة وكتابتها- ولكن، سأوضح المزيد عن هذا فيما بعد. أولاً، أريد أن أشرح الأجزاء الخاصة الأخرى للـ TBarChart.

إن حقول البيانات الخاصة التى يجب أن يتم نشرها تكون عادة مسبوقة بالحرف F. ورغم أن هذا التقليد ليس مطلوباً، إلا أنه يساعدك على إقتفاء أثر

الاسماء الـ publish الخاصة . على سبيل المثال ، إن اسم الخاصية الـ publish للـ FData هى Data . فى برمجة الـ component ، يجب أن تكون قادراً على التمييز بين الاسماء الـ publish والخاصة للحقل .

تعرف الـ TBarChart class حقول الـ FPen ، FBrush ، FData ، و FLabels . قم بتجربة هذه الخصائص فى البرنامج . على سبيل المثال ، اضغط مرتين خاصية الـ Brush فى نافذة Object Inspector ، وقم بتغيير القيمة الفرعية Color لها . وكذلك ، قم بربط خاصية الـ Labels من True و False لمسح labels نقطة البيانات فوق الـ bars .

تعرف أيضاً الـ TBarChart class بعض المتغيرات المتنوعة فقط للملائمة . وبتعريف هذه المتغيرات بعد الوصول إلى القطاع الـ private ، يمكن أن تكون واثقاً من أن أى module أخرى لن تستطيع التأثير على الـ TBarChart objects إلا من خلال الـ methods التى توفرها الـ class بشكل خاص .

بالإضافة الى الحقول ، أيضاً القطاع private بالـ class تعرف خمسة methods : أربعة procedures و function واحدة . تقوم الـ Set procedures بتعيين قيم الى الحقول الخاصة بها . فالـ SetPen يعين قيمة للـ FPen ، والـ SetBrush يعين قيمة للـ FBrush ، الـ publish باستخدام هذه الـ methods لكتابة قيم الحقول المرتبطة بها . وتقوم الـ YData function بإدخال إحداثية عدد صحيح لنقطة بيانات مفهرسة بـ N .

ومرة أخرى نقول أن كل هذه الـ methods تكون private للـ class . هذا يعنى أن العبارات الموجودة فى هذه الـ unit فقط تستطيع أن تستدعيها . ولا يمكن لأى modules أخرى أن تستدعى الـ SetPen أو الـ SetLabels . هذا يقيد استخدام الـ component ويساعد على منع الأخطاء . ولكن ، لأن الخصائص تكون public ، فإن التعيينات إليها باستخدام الـ Object Inspector تؤدى الى استدعاء الـ SetPen والـ SetLabels .

Tip **فكرة:** فى القطاعات الـ public ، والـ protected والـ private للـ class ، يجب أن تسبق تعريفات الحقل تعريفات الـ method والخاصية .

وبعـد القطاع private للـ TBarChart class، تقـوم الـ class بتـعـريف procedure اسـتـبـدالى، وهـو Paint، فى القطاع الـ protected. وترث الـ TBarChart class الـ procedure Paint من الـ class السـالفـة لهـا، ولكى تستـبـدل هـذا الـ method، فـيـجب ان تنهى التـعـريف (بالـ override). ان الـ methods والحـقـول المقـررة فى القطاع الـ protected تسـتطيع الـ unit ان تستخدمها، وكذلك أى classes مشتقة من الـ TBarChart.

ولكن لا يستطيع مستخدمى الـ component استدعاء الـ protected Paint مباشرة. وهذا يوفر وسيلة أمنية لأن الـ Paint يتم استدعاءه استجابة لتلقى النافذة رسالة wm_Paint. وتحل الـ class الجديدة محل الـ Paint بحيث تستطيع رسم bar chart، ولكن سـوف يكون من قبـيل الخطأ أن تسـتـدعى التطبـيقـات هذا الـ procedure مباشرة يمكنك تعريف حقول و methods وخصائص فى القطاع الـ protected للـ class.

وتعـــرف الـ TBarChart class أثنان من الـ method آخـــرين: الـ Constructor (create) والـ destructor (destroy). وكـلاهـمـا مـشـتـقـان، وكلاهما يحتاج الى override. وإنهـمـا معرفان فى القطاع الـ public للـ class بحيث يستطيع المستخدمون استدعـائهمـا. اذا لـم تضع الـ Create فى القطاع الـ public، فلن يكون أمام البرامج سبيلاً لإنشاء الـ TBarChart. يمكنك تعريف حقول و methods وخصائص فى القطاع الـ public. وأى عبارة فى أى مكان من التطبيق الذى يستخدم الـ TBarChart يمكنها استدعاء methods عامة، وقراءة وكتابة أى حقول بيانات عامة.

وأخـيـراً فى الـ TBarChart يوجـد قطاع طويل مـسـبـوق بتـعـريف الـ published. فى هذا القطاع توجـد اسـمـاء الخـصـائص التى تريد من Delphi أن يوفـرها فى نافـذة الـ Object Inspector للـobjects الخـاص بالـ class. وأى methods تعتبر public- مثل الـ StyleChanged، ولكنك غالبـاً مـا تنشر فقط اسماء events وحقول البيانات.

وتعتـبـر بعض خـصـائص الـ published TBarChart جـديدة، وبعـضـهـا موروثاً من class سالفة. على سبيل المثال، ان خاصية الـ Visible متاحة فى كل الـ

البصرية اذا لم تكن تريد ان يقوم مستخدمى الـ components components بتغيير قيمة الـ Visible ، فلا تضعها فى قطاع الـ published . ان تحديد أى الخصائص يجب أن تكون الـ published امر يرجع إليك ، ولكنك غالباً اغلب الخصائص الموروثة فى الـ الـ published، مثل الـ Enabled والـ Visible التى يتوقع المستخدم ان يجدها فى الـ .Object Inspector .

ان الخصائص الجديدة على الـ TBarChart تستخدم نوع مختلف من تعريف الخاصية والذى يحدد مسار الوصول الى البيانات . على سبيل المثال، دقق النظر فى التعريف الـ published لخاصية الـ Pen :

```
property Pen: TPen read FPen write SetPen;
```

فهناك نقطتان بعد اسم الخاصية Pen، بعدهما يوجد تعبير يخبر عن نوع بيانات الخاصية (TPen) ، methods الوصول لقراءتها وكتابتها . وتستطيع عبارات الـ .Object Inspector مباشرة تعيين قيم TPen ، بحيث يكون method الوصول للقراءة هو ببساطة اسم حقل بيانات الـ FPen . ولكن method الكتابة write يعتبر method خاصاً فى الـ SetPen class . هذا يعنى ان كل المواصفات لخاصية الـ Pen تستدعى الـ SetPen بالفعل .

وتستخدم الخصائص الأخرى برمجة مشابهة . بوضعها فى قطاع الـ published وتوفير methods الوصول للقراءة والكتابة ، فإنك تتحكم بحذر فى استخدام بيانات الـ component . ان الـ TBarChart ما هو الامثال بسيط نسبياً، ولكن فى الحالات الأكثر تعقيداً، فإن هذا التحكم لا يقدر بثمن على سبيل المثال، يمكنك برمجة الـ Set method لمنع المستخدمين من تعيين قيم غير مسموح بها أو خارجة عن نطاق للمتغيرات .

وهناك طرق اخرى متاحة للخصائص الـ publish اذا قمت بتوفير method لـ (read) فقط ، فتكون الخاصية read-only . يمكنك ايضاً انشاء خاصية write-only لتقييد الحقول الهامة مثل كلمة المرور ، ولكن هذه الاستخدامات تكون نادرة . ويمكن لعبارات الخاصية ان يكون لها اوامر إضافية للقيم الافتراضية وتحدد ما اذا كان يجب تخزين الخاصية فى ملف form . وهذه التقنيات تعتبر خارج نطاق مقدمة هذا الباب، لكن يمكنك القراءة عنها فى الـ online helpDelphi أو فى الـ Developer Guide .

والسطر الاخير فى واجهة تطبيق الـ Barchart unit يعرف الـ procedure والذى يجب ان يكون لدى الـcomponent units للتثبيت على . الـ paletteويقوم الـ Component Expert بادخـال هذا التـعـريف بدلاً منك ، ولكن اذا كنت component من لا شئ، يجب ان تتذكر ان تدخل هذا السطر :

procedure Register;

يقوم Delphi باستدعاء الـ Register عندما تقوم بتثبيت الـ component على الـ palette والـ unit التى تفتقد الـ Register procedure لا يمكن تثبيتها . يتم استدعاء الـ Register فقط عندما يتم ربط الـcomponent فى الـ palette . لا يجب ان يقوم البرنامج ابداً باستدعاء الـ Register، والذى يزيلة الـ Delphi linker من التطبيقات التى تستخدم الـcomponent module .

: Component class registration

توضح القائمة (٢٠-٣) تنفيذ الـ Register الخاص بالـBarchart unit . يقوم الـ procedure باستدعاء الـ RegisterComponents ، والذى يتطلب arguments : اسم الـ component page ، ومجموعة من الـcomponent classes فى الـ unit ضع اقواساً مربعة حول واحداً أو أكثر من اسماء الـ classes لإنشاء الـ set .

القائمة (٢٠-٣): تنفيذ الـ Register procedure الخاص بالـ Barchart Unit

```
{ Delphi calls this to install component onto the palette }
procedure Register;
begin
  RegisterComponents('Samples', [TBarChart]);
end;
```

اذا كانت الـ unit لها component classes متعددة ، قم بإدخالها فى مجموعة اسم الـ component ، تفصلها الفصلات . على سبيل المثال ، فى unit ذات ثلاثة classes ، يمكنك استدعاء الـ RegisterComponents كهذا :

```
RegisterComponents('Samples', [THisCtrl, THerCtrl,
TYourCtrl]);
```

أو ، يمكنك استدعاء الـ RegisterComponents عدة مرات من الناحية العملية تعتبر هذه العبارات تماماً مثل العبارة السابقة ، ولكن اسهل فى التحرير فى حالة ما اذا اضطررت الى تغيير عدد الـcomponents تسجلها الـ unit . وكذلك استخدم هذه الـform لتثبيت components على pages مختلفة (يجب عليك ان تضع اقواساً حول اسماء لـcomponents) :

```
RegisterComponents('Dialogs', [THisCtrl]);
RegisterComponents('Additional', [THerCtrl]);
RegisterComponents('Samples', [THisCtrl]);
```

لا يجب عليك تسجيل كل class فى unit ، فقط تلك الـclasses فى الـ unit التى تريد تثبيتها على components . قد توفر الـ unit ايضاً procedures ، functions ، ومتغيرات ، وclasses ، و components غير بصرية لا تحتاج ان يتم اختيارها من على اللوحة فى وقت التصميم .

:Component initialization

ان الـ Create constructor الخاص بالـ component يبدأ لـ class object . لكل component يجب أن يكون Create constructor . اذا لم يكن للـ component الخاص بك شيئاً للبدء ، فإنك لا تحتاج الى إبطال الـ Create constructor الذى ترثه من الـ class السالفة ، ولكن نادراً ما يكون هناك component لا يجب عليه أداء بعض العمليات عند الإنشاء .

وبنفس الطريقة ، فإن كل الـ component تقريباً تحتاج الى الـ Destroy لتحرير أى resources للـ component objects . على سبيل المثال ، اذا قمت بإنشاء object أو خصصت بعض الذاكرة فى الـ component constructor ، فيجب أن تضيف deallocation فى الـ destructor ومن المؤكد أن يؤدى هذا إلى فقد نهائى للذاكرة (وهو ما يسمى بخطأ memory leak) . توضح القائمة (٢٠-٤) constructor و destructor الـ TBarChart .

القائمة (٢٠-٤): destructor و constructor الـ TBarChart

```
{ Create component instance at runtime AND design time }
constructor TBarChart.Create(AOwner: TComponent);
```

```
begin
  inherited Create(AOwner);
    Width := 65;
    Height := 65;
    FPen := TPen.Create;
    FPen.OnChange := StyleChanged;
    FBrush := TBrush.Create;
    FBrush.OnChange := StyleChanged;
    FData := TStringList.Create;
    FLabels := True;
end;

{ Destroy component instance at runtime AND design time
}
destructor TBarChart.Destroy;
begin
  FPen.Free;
    FBrush.Free;
    FData.Free;
    inherited Destroy;
end;
```

يجب أن يستدعى الـ Create دائماً الـ constructor الخاص به، عادة مثل الخطوة الأولى. وبخلاف methods الموروثة، يجب أن تقوم باستدعاء الـ Create constructor الموروث لا تغفل هذه الخطوة أبداً، أولاً يتم بدء object بعد استدعاء الـ Create الموروث، يعين الـ constructor قيم لخصائص الـ Width والـ Height، الموروثة من الـ class السالفة. وهذه القيم تحدد المظهر الأول للـ object عندما يضفه المستخدمون إلى الـ form.

بعد ذلك، يقوم الـ constructor بإنشاء حقول الـ FPen فى الـ FBrush والـ FData وجميعها Delphi classes objects ويعتبر هذا مثالاً جيداً لكيفية إستطاعة الـ component unit أن تتوصل الى التعريفات الخاصة لـ class ما. بالإضافة الى إنشاء الحقول، يعين البرنامج procedure وهو الـ StyleChanged،

لـ OnChange event الخاص بالـ FPen والـ FBrush. واستخدام الـ Object Inspector لتعيين قيم لتلك الحقول يؤدى الى استدعاء الـ StyleChanged، والذى يجعل النافذة غير صالحة (جرب هذا مع برنامج). ولهذا السبب، فعندما يقوم المستخدمون بتغيير لون خاصية الـBrush، مثلاً، يكون التأثير فورى.

وكخطوة اخيرة يحدد الـ component قيمة افتراضية لحقل الـ Boolean، FLabels. ويمكن لعبارة الخاصة بالـ published فى تعريف الـ class ان تحدد نفس هذه القيمة لمنع كتابة هذا الحقل الى ملف الـ form اذا كانت قيمة الخاصة ملائمة. ولان constructor بين هذه القيمة، فلا يوجد سبب لهدر مساحة فى كتابة القيمة الافتراضية الى ملف الـ form. للقيام بهذا، قم بتغيير تعريف الخاصية الى مايلى (هذا سطر واحد بالرغم من انه موضح هنا على سطرين):

```
property Labels: Boolean read FLabels write SetLabels
default True;
```

كما ذكرت، يجب ان يقوم الـ destructor الخاص بالـ class بتحرير أى resources. ان الـ destructor الخاص بالـ TBarChart، وهو Destroy، يقوم بهذا العمل باستدعاء Freeلحقول الـ FPen، والـ FBrush ، والـFData التى انشاها الـ constructor وكخطوة اخيرة، يستدعى الـ destructor خاصية الـ Destroy. يجب على الـ Components ان تفعل هذا حتى يتم تحرير أى مواصفات لـ class سالفة. لا تغفل هذه الخطوة ابدا.

وأى exception تحدث فى الـ constructor تترك الـ object وقد تم بدئه جزئياً. يجب ان تبرمج الـ destructor على ان يعالج هذا الاحتمال- على سبيل المثال، بالتأكد من ان الـ pointers قد تم بدؤها تحرير الذاكرة المخصصة لها.

ولايتم فقط استدعاءالـcomponent class الخاص بالـ component فى وقت التشغيل، ولكن ايضاً عندما يقوم المستخدمون بإضافة الـ component object على form. وبنفس الطريقة، اذا قام المستخدم بحذف الـ object أو اغلاق الـ form أو المشروع، يستدعى البرنامج الـ component destructor . ويمكن استدعاء. methods اخرى ايضاً-على سبيل المثال، لعمل مواصفات للخصائص. ان الـ Component objects تكون نشطة فى وقت التشغيل واثناء تطوير التطبيق.

:Visual component painting

لإعطاءها مظاهر رائعـة على مـسـرح Delphi، يجب ان تتـمكن الـ components البصرية من الـ Paint method الموروثه. ان الـ method يستدعى الـ Canvas functions، ويحـدد الـ pen والـ brush color، ويقـوم بكل مـا هو ضرورى لرسم الـ component object .

ان Delphi يهتـم بتوفيـر الـ methods، وميزات الـ clicking والسـحب، واعادة تحديد الحجم، والميزات الاخرى لوقت التصميم. ان كل ما تحتاج إليه هو ان تعرف كيف تريد ان يبدو الـ component object الخاص بك. توضح القائمة (٢٠-٥) تنفـيـذ الـ TBarChart Paint method مع دعـم الثـوابت والـ YData function، والتى تحول نقاط البيانات الى قيم احداثية الـ y لرسم chart bars .

ويقـوم الـ Paint باسـتـدعـاء الـ Windows GDI functions من خـلال خدمات الـ Canvas object الموروث من الـ TGraphicControl. لرسم نافذة الـ BarChart-وهو اول مايراه المستخدم عند إضافة الـ BarChart على form-يقوم الـ Paint بتعيين الوان الـ Pen والـ Brush الخاصة بالـ Canvas باستخدام حقول البيانات FPen و FBrush، والتى يستطيع المستخدم برمجتها فى الـ Object Inspector. بعـد اعـداد القليل من المتـغـيـرات، يقـوم الـ Paint باسـتـدعـاء الـ Rectangle، وكالسحر، يظهر مربع ابيض فى الـ form. هذا هو كل ما تحتاج ان تفعله لتعطى الـ components المظهر الخاص بها فى وقت التصميم.

القائمة (٢٠-٥): تنفيذ الـ Paint method الخاص بالـ TBarChart

```
const
{ Fixed constants }
 numClrs = 16;          { Number of colors in colorArray }
   spaceAtBottom = 10;    { Reserved pixels below chart
  }
   spaceAtLeft = 20;      { Reserved pixels at left of chart
  }
   spaceAtTop = 40;       { Reserved pixels above chart }
   spaceAtRight = 20;     { Reserved pixels at right of
  chart }
```

```
   yScaleMax = 100.0;     { Maximum Y scale value }
    yScaleIncrement = 10.0; { Increment for Y scale
   markers }
{ Typed constants }
 spaceVertical: Integer = spaceAtTop + spaceAtBottom;
    spaceHorizontal: Integer = spaceAtLeft +
   spaceAtRight;
    yScale: Integer = Trunc(yScaleMax / yScaleIncrement);
{ Array of colors used to draw topmost bars }
 colorArray: array[0 .. numClrs _ 1] of TColor = (
     $0000000, $0FFFFFF, $0FF0000, $000FF00,
       $00000FF, $0FFFF00, $000FFFF, $0FF00FF,
       $0880000, $0008800, $0000088, $0888800,
       $0008888, $0880088, $0448844, $0884488
      );

{ Return Y-coordinate for data point N }
function TBarChart.YData (N: Integer): Integer;
var
 F: Double;
begin
 F := (StrToFloat (FDat [N]) / yScaleIncrement) *
    yIncrement;
    Result := YBase _ Round(F);
end;

{ Paint component shape at runtime AND design time }
procedure TBarChart.Paint;
var
 XMax, YMax: Integer;
    Width1, WidthD2: Integer;
    I, X1, Y1, X2, Y2: Integer;
begin
 with Canvas do
    begin
    { Erase background }
```

```
Pen.Color := FPen.Color;
   Brush.Color := FBrush.Color;
   X1 := Pen.Width div 2;
   Y1 := X1;
   XMax := Width _ Pen.Width + 1;
   YMax := Height _ Pen.Width + 1;
   Rectangle(X1, Y1, X1 + XMax, Y1 + YMax);
   if FData.Count = 0 then Exit;
  { Initialize variables }
try
    XIncrement := (XMax _ spaceHorizontal) div
   FData.Count;
      YIncrement := (YMax _ spaceVertical) div
   yScale;
      Width1 := XIncrement div 2;
      WidthD2 := Width1 div 2;
      XBase := spaceAtLeft + WidthD2;
      YBase := YMax _ spaceAtBottom;
      Canvas.Font := Self.Font;
 { Draw barchart }
    for I := 0 to FData.Count _ 1 do
      begin
X1 := spaceAtLeft + (XIncrement * I);
        Y1 := YData(I);
        X2 := X1 + Width1;
        Y2 := YBase;
        if FLabels then
        begin
          Brush.Color := FBrush.Color;
          TextOut(X1, Y1 _ 30, FData.Strings[I]);
end;
        Brush.Color := clBlack;
        Rectangle(X1 + 4, Y1 _ 4, X2 + 4, Y2 _ 4);
        Brush.Color := colorArray[(I + 2) mod
     numClrs];
        Rectangle(X1, Y1, X2, Y2);
```

```
          end;
            except
        ShowMessage('Error in data point ' + IntToStr(I));
              FData.Clear;
              Invalidate;
            end;
          end;
      end;
```

وبالطبع ، يحتاج الـ component الخاص بك أن يقوم بما هو أكثر من مجرد عـرض مـربع بسيط . إن باقى الـ TBarChart Paint method يرسم للـ bars و text labels (اذا كـانت الـ FLabels مـحـددة بـTrue) . وهذا الجزء من البـرنامج يغطى عنصرين هامين . الأول ، اذا كانت الـ FData.Count مساوية للـصفر ، فلا يوجد نقاط بيانات فى قائمة الـ string لخاصية الـ Data . وينتهى الـ Paint فى هذه الحالة ليمنع خطأ القسمة على صفر . والجانب الثانى هو معالجة الـ exception ، والذى يوفره الـ Paint بإدخال أوامر الرسم والحساب الخاصة به فى try block .

وبالرغم من هذا المستوى من الحماية ، فإن أى exceptions تحدث فى الـ Paint method يمكن أن تؤدى الى loop لا نهائية- حالة لا قياسية بأن الـ TBarChart تعمل فى الـ except block الخاص بها . اذا وقع خطأ فى الـ try block ، يعرض Delphi الـ displays والذى عندما يكون مغلقاً ، يجعل النافذة السفلية تصبح غير صالحة ، مما يجـعل الـ Windows يصـدر رسـالة wm_Paint أخرى ، مما يعـيد إدخـال الـ Paint method ، والذى يولد exception آخر ، وهكذا الى ما لا نهاية .

لهـذا السبب ، تعالج الـ TBarChart أى exceptions بعرض رسالة خطأ ، أو مـسح الـ FData string list ، وإدخـال صـلاحـية النافـذة . جـرب هذا بإدخـال بعض البيـانات مثل XYZ أو string آخر فى خاصية الـ Data . سوف تتلقى رسالة خطأ تخبرك بفهرس نقطة البيانات الخطأ (إن الفهرس الأول هو صفر كما هو الحال فى كل string lists) ، ويصبح الـ object خالياً .

وبدلاً من الـ clear Data ، وهوغـالبـاً ليس أفـضل اسـتـجـابة ، قـد يعـرض البرنامج رسالة أو رمز داخل الـ object ، أو يمكن أن يحدد علماً يمكن للبرنامج أن يفحصه . قد تريد تجربة هذه الاستجابات البديلة فى الـ except block .

:component property access methods

توضح القائمة (٢٠-٦) باقي برمجة الـ TBarChart. وهذه الـ procedures تنفذ methods وصول الخاصية لحقول الـ FPen، FBrush، FData و FLabels.

القائمة (٢٠-٦): تنفيذ method وصول لخاصية الـ TBarChart

```
{ Local event handler redraws shape when necessary }
procedure TBarChart.StyleChanged(Sender: TObject);
begin
  Invalidate;
end;

{ Assign new brush data to FBrush field }
procedure TBarChart.SetBrush(Value: TBrush);
begin
  FBrush.Assign(Value);
end;

{ Assign new pen data to FPen field }
procedure TBarChart.SetPen(Value: TPen);
begin
  FPen.Assign(Value);
end;

{ Assign new string list to FData field }
procedure TBarChart.SetData(Value: TStrings);
begin
  FData.Assign(Value);
    Invalidate;
end;

{ Assign new Boolean value to FLabels field } .
procedure TBarChart.SetLabels(Value: Boolean);
begin
```

```
        if FLabels <> Value then  { Exit if no change needed }
          begin
            FLabels := Value; {Assign to FLabels NOT Labels
          !}
            Invalidate;{Redraw component to add/remove
          labels}
        end;
      end;

      end.
```

إن الـ StyleChanged procedure يستدعى الـ Invalidate لذا يصدر الـ Windows رسالة wm_Paint لنافذة الـ object . هذا يؤدى فى النهاية الى استدعاء لل Paint والذى يمحو ويعيد رسم صورة الـ bar chart . إن البرنامج لا يقوم باستدعاء الـ StyleChanged ؛ فهو يعين الـ procedure للـ OnChange للـ TPen والـ TBrush . ولذلك ، فإن أية تغييرات لخصائص الـ Pen والـ Brush تؤدى لإعادة رسم الـ object .

وتعمل الـ Procedures SetBrush والـ SetPen والـ SetData بطريقة متشابهة . فكلاً منها يتلقى parameter من نوع الـ field والذى يقومون بتمريره لل Assign method للحقل . يمكنك أن تثق من أن الـ Assign يفعل كل ما هو ضرورى للتخلص من أى resources موجودة قبل قبول البيانات الجديدة . على سبيل المثال ، إن تعيين Pen جديد يتخلص من الـ pen resource للـ Windows (إن وجد) المشار إليه بواسطة حقل الـ FPen .

لاحظ أن الـ SetData ، الذى يعين خاصية الـ Data string list ، يقوم باستدعاء الـ Invalidate لإعادة رسم النافذة . وهذا لأن الـ TStrings class لا توفر OnChange ؛ واذا كانت توفره ، يمكنك تعيين الـ StyleChanged كما كان الحال مع خصائص الـ Pen والـ Brush .

فى method الوصول لخاصية ، يمكن أن يساعد التأكد مما اذا كانت قيمة ما هى نفسها كالقيمة الجديدة على منع الـ flicker ، وخاصة للـ components البصرية المعقدة مثل الـ BarChart . اذا قامت الـ SetLabels بتعيين القيمة الجديدة واستدعت الـ Invalidate ، فسوف يحدث للـ object الـ flutter فى كل مرة يقوم المستخدمون فيها بإبراز خاصية الـ Labels فى الـ Object Inspector .

تعين الـ SetLabels قيمة مستخدم لمفتاح الـ Boolean للـ FLabel . وهذا الـ procedure يوضح تصـمـيـمـاً عـامـاً لـ methods الوصـول لخـاصـيـة . وطبـقـاً للمعروف ، تقوم الـ method أولاً بالتأكيد من أن الـ Value هى نفسها مثل حقل الـ object . اذا كانت كذلك ، فـلا يمكن فـعل شئ ، وينتـهى الـ procedure . وإن لم تكن ، يعين البرنامج Value لحقل الـ FLabels ، ويبطل صـلاحية النافـذة لإعادة رسمها وإزالة أو إدخال labels نقاط البيانات .

لا تقم ابداً بتعيين قيم لخصائص فى method الوصـول لخاصية . وعين قيم فقط لحقول البيانات ، والمعرفة غالباً فى القطاع الـ private فى للـ class . ان تجاهل هذه القـــاعـــدة يمكن ان يؤدى . الى زيادة المخـــزون ، وقـــد يغلقDelphi والـ Windows . فى القائمة النموذجية ، تعتبر الـ SetLabels هى method الوصول الذى يستدعيه البرنامج للتعليمات لخاصية الـ Labels . اذا كانت الـ SetLabels لتـعـيين قـيـم للـ Labels ، ولكان البـرنامج قـد اسـتـدعى فى الـ SetLabels مـراراً وتكراراً الى ان ينفجر المخزون . كن شديد الحذر عند كتابة method وصول لخاصية لتعيين قيم لحقول بيانات فعلية فقط فى الـ class .

Caution

تحذير: إن تعيين قيم لخصائص فى method الوصول لخاصية يمكن أن يؤدى الى مـا يسـمى بالتكرار اللا نهـائى ، مما يؤدى الى زيـادة المخزون . وبالطبع ، إن التكرار لا يكون أبدى لأنه يؤدى الى إيقـاف النظام ، ولكن لا عليك بالتسمية فقط لا تفعل هذا! .

فهم تصميم الـComponent:

إن ما تبـقى من هذا البـاب يقدم أفكاراً تساعد على فـهم تقنيات تصـمـيم الـ component . كـما ذكـرت ، إن إنشـاء الـ component موضـوع قـد يملأ كتاباً بهـذا الحـجـم ، لذا فإن مـا يلى لا يعتـبر كـاملاً ووافـيـاً . ولكن ، لـقـد حـاولـت ان اتلمس الموضـوعات التى توليـها اهتـمـامـاً رئيسـياً كـمؤلف للـ component ، والتى تقتـرح موضوعات للبحث فى مصادر اخرى .

الـCustom components:

فى الـ VCL ، تمر على components تحمل الكلمة Custom فى أسمائها . على سـبـيل المثـال ، ان الـ TCustomEdit مـشـتق من الـ TWinControl فى الـ StdCtrls unit باستخدام هذا التعريف :

```
TCustomEdit = class(TWinControl)
...
end;
```

وبعد ذلك مباشرة، تجد هذا التعريف للـ TEdit component، وهو الـ component الذى يستخدمة التطبيق :

```
TEdit = class(TCustomEdit)
...
end;
```

والسبب فى وجود two classes بدلاً من مجرد الـ TEdit هى ان نزودك بـ raw edit-control class بلا خصائص الـ published (فى الواقع، هناك خاصية واحدة، وهى الـ TabStop، والتى لها قيمة افتراضية هى True) والـ Components التى تحمل كلمة Custom فى اسمائها لا تجعل الخصائص published. وقد تحتوى ايضاً على تعريفات للـ abstract method والتى تتوقع الـ class المشتقة تنفيذها.

اذا قمت باشتقاق الـ TEdit، فإن كل الخصائص الـ published لها تصبح published فى الـ class المشتقة. واذا قمت، بدلاً من ذلك، باشتقاق class من TCustomEdit، والتى توفر البرمجة الفعلية للـ TEdit، فإنك تحدد أى الخصائص يتم نشرها. وتنفذ الـ class المشتقة ايضاً أى .abstract methods.

:Component debugging

ان إزالة الاخطاء واختبار الـ components يصبح امراً معقداً بسبب انك تستخدم نفس نظام التطوير الذى تريد ان تثبت الـ components فيه. يستطيع Delphi ان يقوم بعملية الـ compile فقط لمشروعات لإزالة الاخطاء-لإزالة اخطاء component يجب ان تثبتها على الـ VCL وتكتب تطبيقاً اختبارياً. عندما لا يكون هذا ملائماً، يمكنك ان تنتهج أحد اسلوبين :

* قم بتطوير الـ component على انه تطبيق، ثم قم بتحويله الى component. لقد انشأت الـ TBarChart باستخدام هذا الـ method. ان البرنامج الاصلى، وهو غير مذكور هنا، قد عرض bar chart باستخدام class

مـشـتـقـة من الـ TGraphicControl . ثم قـمـت بنسخ هذه الـ code فى الـ component unit. النهائية .

* قم بضم .component unit فى مشروع اختبارى، وقم بانشاء objects من الـ component class فى procedure أو function . يجب عليك ان تقـوم بتشغيل البـرنامج لترى الـ component، ولكن هذه الطريقة تساعدك على إزالة الاخطاء من الـ class قبل تثبيت الـ component على الـ palette .

على القرص المدمج: لتجربة التقنية الثانية- والتى تعتبر ناجحة فى إنشاء واختبار الـ component المعقدة- انسخ ملف الـ BarChart.pas من على دليل الـ Source\BarChart بالقـرص المدمج الى دليل جـديد وخالى. اتبع هذه الخطوات لإنشاء برنامج اختيارى ينشئ BarChart تحت التحكم الكامل للبرنامج. توضح القائمة (٢٠-٧) كيفية إنشاء object فى وقت التشغيل لـ component ليس موجوداً على لوحة الـ component .

١- إبدأ تطبيـقـاً جـديداً. إجـعل اسـم الـ form MainForm، وحـدد الـ Caption الخاص بها بـ Test BarChart Component. احفظ المشروع فى نفس الدليل الذى يحتوى على نسخة من الـ BarChart.pas . (من الناحية العـملية قد تختلف الأدلة، ولكن من الأسـهل أن تسـتخدم نفس الدليل لهذا العرض). اجعل اسم الـ unit Main.pas واسـم المشروع BarTest.prj .

٢- اضف BarChart الى أمر الـ uses بالـ Main unit .

٣- بدلاً من اسـتـخـدام الـ BarChart على الـ VCL (عند اخـتـبـار component جديد، فهذا لن يكون ممكناً)، قم بتعريف object من الـ BarChart class يدوياً. افعل هذا بكتابة تعريفات الـ object فى الـ form class- تماماً كما يفعل Delphi بصورة تلقائى لـ components التامة. على سبيل المثال، أدخل التعريف التالى فى القطاع الـ public لـ TMainForm class :

```
TMainForm = class(TForm)
private
  { Private declarations }
public
```

```
    BarChart1: TBarChart;
  end;
```

٤- قم بإنشاء الـ OnCreate الخاص بالـ form (أو يمكنك استخدام أى OnClick -event handler للـ Button، مثلاً). قم بانشاء component instances التى تحتاجها . هذا يتطلب على الاقل خطوتين : (١) استدع Create لإنشاء الـ object، و (٢) عين parent object لحقل الـ parent object. وغالباً، تستطيع ان تستخدم الـ Self كـ arguments فى كلتا المهمتين. وتشير الـ Self الى الـ form object (لان هذا يعد method للـ form class)، وإنك غالباً ما تريد ان تمتلك الـ form الـ object الناتج من الخطوة رقم (١)) وتكون الاساس لها [الخطوة رقم (٢)]. على سبيل المثال، استخدم هذه العبارات لإنشاء BarChart :

```
BarChart1 := TBarChart.Create(Self);
BarChart1.Parent := Self;
```

٥- يجب عليك ايضاً ان تبدأ حقول اخرى مثل Left ، Right ، Width، و Height والتى تحدد موضع الـ object وحجمه. بالإضافة الى ذلك، إنك تحتاج لـ إضافة عبارات لتعيين أيه بيانات مطلوبة-قائمة الـ Data string للـ BarChart، مثلاً. توضح القائمة (٢٠-٧) الـ unit التامة، والمخزنة على انها Main2.pas فى دليل الـ Source\BarChart على القرص المدمج. ويوضح هذا الملف كيفية إنشاء object فى وقت construct ليس على VCL . استخدم هذا الملف لإنشاء مشروع اختبارى للـ BarChart class .

القائمة (٢٠-٧): Barchart\Main2.Pas. استخدم هذا الملف لإنشاء مشروع اختبارى للـ BarChart class

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics,
   Controls,
    Forms, Dialogs, BarChart;
```



```
type
 TMainForm = class(TForm)
     procedure FormCreate(Sender: TObject);
      private
     { Private declarations }
      public
     BarChart1: TBarChart;
      end;

var
 MainForm: TMainForm;

implementation

{$R *.DFM}

procedure TMainForm.FormCreate(Sender: TObject);
begin
 BarChart1 := TBarChart.Create(Self);
    BarChart1.Parent := Self;
    with BarChart1 do
    begin
     Left := 20;
        Top := 20;
        Width := 375;
        Height := 200;
        Data.Add('65.0');
        Data.Add('45.0');
        Data.Add('95.0');
        Data.Add('76.0');
        Data.Add('51.0');
        Data.Add('90.0');
      end;
end;

end.
```

لا تقم بادخال حقول بيانات أو تعريفات للـ method فى قطاع الـ published الافتراضية للـ form class (ان القطاع التالى مباشرة لتعريف الـclass حتى محدد الوصول، وغالباً يكون القطاع private). اذا قمت بتعديل القطاع الـpublished الافتراضى، فإنك تتلقى رسالة خطأ لأنه هذا هو المكان الذى يعرف فيه Delphi الـ objects والـ event handlers.

خصائص الـ Class:

ان الخاصية قد تكون أى نوع بيانات فيما عدا الملف. ولكن، لا يمكن للخاصية ان تكون Pascal array، لان الخصائص ليست حقول حقيقية والوصول إليها محكوم بـmethods القراءة والكتابة. ولكن، يمكنك انشاء خصائص شبيهة بالـ array التى تستدعى methods لقراءة وكتابة عناصرها. (لمزيد من المعلومات، انظر خصائص الـ array فى هذا الباب). باستثناء هذه القيود، قد تنشر الخاصية أى نوع من الحقول.

ان الـ events تعد حقول بيانات فعلية للـTNotifyEvent أو نوع مشابه. ان الـ event هو pointer لـ method تم توفيره من قبل المستخدم. ولكن الـevent فى حد ذاتهmethod، ولذلك، فإن نشر الـ events يؤدى الى نشر حقل بيانات (الـ pointer)، ليس code.

تظهر الخصائص لمستخدم الـ component كحقول، ولكنها methods مغلقة لقراءة وكتابة حقول بيانات اخرى، والتى تعتبر خاصية للـcomponent class. ويمكن ايضاً للخصائص ان تكون محسوبة، على سبيل المثال، يمكنك كتابة method الوصول لخاصية وتحسب قيمة الخاصية من حقول بيانات اخرى. ولكن، كن على حذر من ان تقدم الكثير من هذه الـ methods فيما بين الخصائص المحسوبة، أو يصبح ترتيب بدءها فى وضع حساس.

ان نشر الخصائص يجعل معلومات وقت التشغيل متاحة عن الـ class التى تستخدمها Delphi بعدة طرق. على سبيل المثال، يستخدم الـ Object Inspector معلومات نوع لعرض وتحرير قيم الخاصية، وهى مخزنة فى ملف dfm.. بعد compile المشروع الـ BarTest لهذا الباب، افتح ملف Main.dfm لتستعرض كيف يقوم Delphi بتحزين الخصائص. توضح القائمة (٢٠-٨) ملفاً. استخدم Delphi لفتح الملف لفحص الخصائص.

القائمة (٢٠-٨): ملف الـ Barchart\Main.dfm لمشروع الـ BarTest

```
object MainForm: TMainForm
 Left = 200
    Top = 95
    Width = 435
    Height = 300
...
 object BarChart1: TBarChart
      Left = 24
         Top = 16
         Width = 377
         Height = 209
         Hint = 'BarChart component'
         Brush.Color = clSilver
         Data.Strings = (
           '65.0'
              '45.0'
...
     '90.0')
            Labels = True
         ParentShowHint = False
         ShowHint = True
       end
end
```

هذا يبـدو كـالـ Pascal، ولكنه ليس كـذلك. انه مـلف يحـتـوى على قـيم الخواص. إن ملف الـ form يحتوى على قيم خاصية كتلك التى تدخل فى نافذة الـ Object Inspector. يقـوم Delphi بتـحـميل قـيم الخـاصـيـة بعـد أن يسـتـدعى constructor الـ component، والذى يقوم ببدء قيم الخاصية.

إن قيم الخاصية يمكن إنشائها بأكثر من طريقة. بعد استدعاء الـ component constructor وتحـميل قـيم الخـاصـيـة من ملف الـ form (أو صـورته المرتبطة بملف code الـ exe.)، يقـوم Delphi باسـتـدعـاء Loaded method ظاهرى والذى يمكنك السيطرة عليه. ضع هذا التعريف فى القطاع الـ protected للـ component class.



```
procedure Loaded; override;
```

قم بتنفيـذ الـ procedure لأداء أى إنشـاءات إضـافـيـة للـ object . وهذه البدايات تأخذ الأولوية عن أى قيم معينة فى الـ constructor أو من ملف الـ form (بمعنى آخر، بواسطة الـ Object Inspector) . لا يزال المستخدمون قادرون على تحديد خصائص باستخدام الـ Object Inspector، ولكن عندما يقوموا بتشغيل البرنامج، يتم استدعـاء الـ Loaded مرة أخرى، لذا لا تستبـدل أى قيم تريد أن يتوصل إليها المستخدمون .

ويتعـبر الـ Loaded مفـيداً فى توفير قيم إفتراضه للخصائص التى يقرر المستخدمون أن يتركوها فارغة أو التى ليس لها قيم ملف خاصية . على سبيل المثال الـ Loaded function، إن الـ DBNavigator يستـخـدم هذه التـقنية لإنشـاء hint-text strings إشارة إفتراضية اذا لم يدخل المستخدم أى strings فى حقل الـ Hints .

استـدع دائمـاً الـ Loaded method الموروث فى procedure الاستبـدال الخاص بك، ويعتبر الـ Loaded أيضاً هو الفرصة الوحيدة التى تحصل عليها للوصول الى أى objects ملحقة خلال عملية التحميل . على سبيل المثال، لا يستطيع الـ data- ware component استخدام خاصية الـ DataSource التابعة له حتى يتم استدعاء الـ Loaded .

Note

ملحوظة: إن الـ Loaded method لا يعد جزءاً من خطوات إنشاء الـ component ولهـذا السبب لا يجب أن تضع code هامة فى Loaded method مـوروث لأنه لا يتم استـدعـاؤه اذا كـان الـ component يتم إنشاؤه ديناميكياً فى وقت التشغيل بدلاً من تحميل الـ from كـ form stream .

خصائص الـ Array:

من المـمكن ألا تكون الخـصـائص Pascal arrays، ولكن يمكنك انشـاء خاصية array يتم استخدامها تماماً مثل الـ array . وتعتبر خصائص الـ array قيمة وبخاصية فى إنشاء هياكل بيانات متقدمة مثل الـ arrays المرتبطة والموزعة . ومثل جميع الخصائص، توفر خاصية المتجةmethods وصول لقراءة وكتابة البيانات .

ويمكن لـ class الخاصية ان تختار أى method للتخزين الفعلى البيانات . على سبيل المثال ، هذه هى الطريقة التى توفر بها خاصية الـ TStrings وصولاً لبيانات الـ string باستخدام فهرسة الـ array ، ولكن داخلياً ، تربط الـ strings ببراعة فى list .

وقد تستخدم خاصيةالـ array ايضاً أى نوع من نوع البيانات للفهارس . يمكنك إنشاء array مفهرس على strings أو قيم النقطة القائمة . ان الـ Pascal arrays العامة قد تستخدم فقط فهارس تركيبة مثل الاعداد الصحيحة ، الرموز ، والثوابت المحدودة .

On the CD-ROM

على القرص المدمج: ان القائمة (٢٠-٩) هى تطبيق DOS-prompt الذى يوضح كيفية برمجة خصائص الـ array . يمكنك ايضاً استخدام نفس التقنية فى الـ component ، ولكن كما يوضح هذا البرنامج ، يمكنك ايضاً نشر خصائص الـ array على انها اعضاء protected لأى class ويوجد هذا البرنامج على القرص المدمج فى دليل Source\Daynames ، فى ملف ArrayP.pas . لتشغيل البرنامج ، اضغط زر الـ Windows Start ، واختر Programs|MS DOS Prompt ؛ ثم تحول الى دليل الـ Daynames ، وإدخل Arrayp .

القائمة (٢٠-٩) : Daynames\ArrayP.pas

```
program ArrayProperties;

uses SysUtils;

type

 TDayNames = class
   private
    function GetName(N: Integer): String;
   protected
    property DayStr[N: Integer]: String read GetName;
  default;
   end;
```

```
function TDayNames.GetName(N: Integer): String;
begin
 if (N < 0) or (N > 6) then
      raise ERangeError.Create('Array index out of range');
        case N of
      0: Result := 'Sunday';
          1: Result := 'Monday';
          2: Result := 'Tuesday';
          3: Result := 'Wednesday';
          4: Result := 'Thursday';
          5: Result := 'Friday';
          6: Result := 'Saturday';
        end;
end;

var
 DayNames: TDayNames;
     I: Integer;

begin
 DayNames := TDayNames.Create;
     try
       Writeln('Default property (DayNames[I])');
           for I := 0 to 6 do
             Writeln(DayNames[I]);
           Writeln;
       Writeln('Named property (DayNames.DayStr[I])');
       for I := 6 downto 0 do
         Writeln(DayNames.DayStr[I]);
         finally
   DayNames.Free;
         end;
end.
```

ملحوظة: لمزيد من المعلومات عن استخدام الـ DOS-prompt compiler خط الأمر لـDelphi، انظر الباب الحادى والعشرين.

ينشئ البرنامج class صغيرة، وهى TDayNames، والتى تقدم ايام الاسبوع فى شكل pseudo-string array. الـ claas تعرف function خاصة، وهى الـ GetName، والتى يتم استخدامها على انها method وصول للقراءة للـ pseudo-string array. ويعرف القطاع الـ protected للـ class لـ method الوصول لخاصية، DayStr، متبوعاً باقواس مربعة تحتوى على argument عدد صحيح. وهذه الـ argument يمكن ان يكون أى نوع بيانات، ولو حتى string. ويكون قوس الإغلاق متبوعاً بنقطتين، نوع بيانات عناصر الـ array، و method الوصول للقراءة، وهو GetName.

ان إنهاء تعريف الخاصية بـ default يحدد هذه الخاصية على انها الخاصية الافتراضية للـ class. هذا يعنى ان المستخدمين لـ class objects يمكن ان يعاملوا الـ objects على انه الخاصية. ويمكن ان يكون خاصية واحدة فقط لأى class هى البديل الافتراضى. على سبيل المثال، ان الـ for loop الاولى للبرنامج تستخدم الـ DayNames object وكأنه array:

```
for I := 0 to 6 do
  Writeln(DayNames[I]);
```

يمكنك ايضاً الإشارة الى الخاصية على انها حقل فى الـ object:

```
for I := 6 downto 0 do
  Writeln(DayNames.DayStr[I]);
```

وتستخدم الـ TStrings class نفس هذا الـ method لتوفير وصولاً الى strings باستخدام تعبير مثل Items[I] أو Items.Strings[I]، وهما نفس الشئ لان الـ Strings هى الخاصية الافتراضية للـ TStrings.

عندما يستخدم البرنامج تعبيراً مثل DayNames[I]، يولد الـcompiler إستدعاءاً للـ method الوصول الى القراءة المحدد، وهو فى هذا المثال، الـ function GetName. وفى القائمة، تقوم الـ GetName اولاً بالتأكيد مما اذا كان



الفهرس فى النطاق- اذا لم يكن كذلك، فإنها تحدث ERangeError exception
إدخل هذه العبارة فى الـ try block للبرنامج الرئيسى لأختبار الـexception :

```
Writeln(DayNames[7]);  { ??? }
```

وتستخدم الـ Function GetNames عبارة case لإدخال string اسم يوم لقيم الفهرس من صفر الى ستة ولان الـ class للقراءة فقط، فإنها توفر method وصول للكتابة، ولكن هذا امر يسهل القيام به. قم بتعريف method وصول الى الكتابة، والذى يجب ان يكون له parameter قيمة فهرس (يمكن ان يكون أى نوع، ولكن هنا فهو Integer)، كما يلى :

```
procedure SetName(N: Integer; const S: String);
```

بعد ذلك، اضف write SetName الى تعريف الخاصية :

```
property DayStr[N: Integer]: String read GetName write
SetName; default;
```

قم بتنفيذ الـ SetName procedure لتخزين string الذى تم تمريره فى الفهرس المحدد. على سبيل المثال، يمكنك إدخال الـ string فى قائمة وتسجيل قيمة الفهرس فى object ما. سوف يبحث method الوصول الى القراءة فى القائمة عن هذا الفهرس ويدخل الـ string المرتبط به- وهذا يعد مثالاً على الـ array المرتبط والموزع ان الـ string المدخلة فقط هى التى يتم تخزينها على قائمة، ومواضع الفهرس الغير مستخدمة فلا تشغل مساحة.

والـ non components classes مثل الـ TDayNames يمكن الا تنشئ خصائص، ولكن يمكن أن تعرف فى قطاع protected أو public كما هو موضح هنا. وهذه تقنية مفيدة فى التحكم فى الوصول الى حقول البيانات.

إنشاء الـActiveX Controls:

يمكن تحويل أى Delphi component بسهولة الى ActiveX control. وهذا الـ control يمكن استخدامه عندئذ فى تطوير نظم مثل الـ Visual Basic، Microsoft Access، Paradox للـ Windows، والبرمجيات الأخرى التى تؤيد بروتوكول الـ ActiveX. يمكنك ايضاً استخدام الـ ActiveX الخاصة بك فى

Delphi وفى الـ C++ Builder الخــاص بالـ Borland، بالرغم من انه اذا كــان لديك الـ control الأصلى كـ component، فـإن افـضل اسـتـخـدام يكون فـى الـ form الأصلية فى Delphi .

كـ ActiveX control، فـإن Delphi component يكون مــحــاطاً فى غلاف الذى يحكم الاتصال بين الـ control والعالم الخارجى . ومواجهة التطبيق هذه تتفق مع تخصيصات الـ ActiveX للـ Microsoft، وهى لا تختلف عن أى ActiveX آخر . ولكن، الـ control يحتوى الـ Delphi component داخله .

كيف تنشئ الـ ActiveX:

لإنشاء ActiveX هناك خطوات عديدة :

١- قم بتصميم واختيار component كما هو موضح فى هذا الباب .

٢- قم بتشغيل الـ ActiveX Control Wizard الخاص بـ Delphi لتحول الـ component الى ActiveX control .

٣- واختيارياً، قم بتشغيل الـ ActiveX Property Page Wizard . هذه الخطوة تؤدى الى إنشاء صفحة خاصية والتى تمكن المستخدمين من استعراض وتحرير خصائص الـ control والتى تعتبر ضرورية اذا كان كل ما تريده هو نشر الـ control على سبيل المثال، المتصفح الـ Internet . اذا اخذت هذه الخطوة، فإنك ايضاً تربط صفحة الخاصية بالـ ActiveX control .

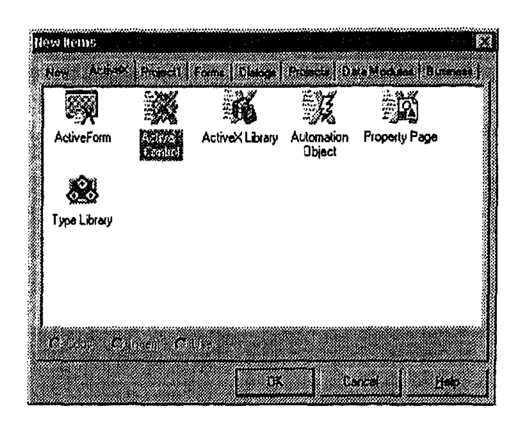
٤- بعد الـ compil يتم تسجيل الـ control مع الـ Windows . يجب ان يتم تسجيل كل الـ ActiveX قبل ان يتم استخدامها .

عندما تقـوم بإنشـاء ActiveX من component، يقـوم Delphi بصـورة تلقـائيـة بإنشـاء type librar والتى تصـــف واجـهـة تطبيق الـ control . وهذه المكتبة، والمخزنة فى ملـف tlb.، تستخدمها نظم التطوير للوصول الى خصائص الـ control، والـ methods، والـ events . تحتـوى المكتبـة أى control آخـر أو ملف code يسـتـخـدم الـ ActiveX عندمـا تقـوم بـ compile هذا الـ control أو التطبيق .

تحويل الـ DingButton الى ActiveX:

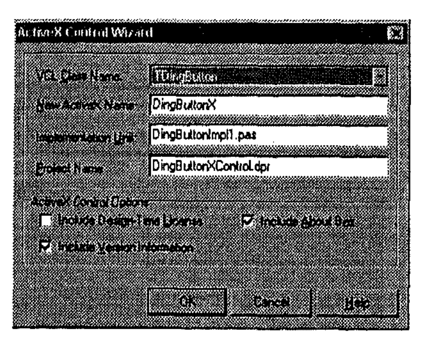
إتبع هذه الخطوات لتحويل الـDingButton الخاص بهذا الباب الى : ActiveX

١- اختر FilelNew... واختر باب صفحة الـ ActiveX من الـ New Items dialog [انظر شكل (٢٠-٣)].



شكل (٢٠-٣): هذه هى الـ ActiveX فى الـ New Items dialog

٢- اضغط مرتين ايقونة الـ ActiveX Control. هذا يؤدى الى عرض الـ ActiveX Control Wizard dialog الموضح فى شكل (٢٠-٤).

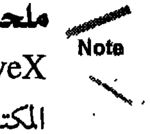


شكل (٢٠-٤): اختر الـ component لتحويله الى ActiveX

٣- اختر الـ component لتحويله الى ActiveX من على قائمة الـ VCL Class Name. عندما تفعل هذا، يملأ Delphi الحقول الاخرى، والتى يمكنك تحريرها اذا اردت. على سبيل المثال، يمكنك إدخال ActiveX Name



وImplementation Unit مختلف، والذى يحتوى على الـsource code لغلاف واجهة تطبيق الـActiveX الخاص بالـcomponent . اضغط Ok عندما تنتهى .



ملحوظة: ان الـActiveX يجب ان تصنف كجزء من مشروع مكتبة الـActiveX . فى هذه النقطة، يحذرك Delphi اذا لم يكن لديك مثل هذه المكتبة مفتوحة- اجب بـYes لإنشاء مكتبة جديدة وأكمل العمل .

ان الخطوات (٢) و (٣) تقدم ثلاثة اختيارات يمكنك تشغيلها (انظر check boxes فى اسفل نافذة الـdialog) . وهذه الخيارات هى :

- **Include Design-Time License**: وهذا يؤدى الى إنشاء ملف lic له مفتاح مطلوب لفتح الـcontrol للاستخدام فى بيئة تصميم البرمجيات . تأكد من هذا الخيار اذا كنت تريد ان تقيد استخدام الـcontrol الخاص بك بالنسبة للمستخدمين النهائيين (كما هو الحال فى متصفح الـInternet)، ولكنك لا تريد مطورى البرمجيات أن يحتوى الـcontrol فى برامجهم دون الحصول على رخصة وقت التصميم منك .

- **Include Version Information**: هذا يضيف معلومات نسخة لملف الـocx. الخاص بالـcontrol . يستطيع المستخدمون ان يروا هذه المعلومات . لإدخال معلومات نسختك، اختر ...Project|Options واختر باب صفحة الـVersionInfo .

- **Include About Box**: هذا يؤدى الى إنشاء الـabout-box dialog unit، والتى يتم إضافتها لمشروع مكتبة الـActiveX . يمكنك تحرير الـabout-box dialog هذا كما تفعل مع أى Delphi form أخرى . على سبيل المثال، اختر الـabout-box dialog فى نافذة الـcode editor واضغط F12 لإظهار الـform البصرية المرتبطة به . يمكنك عندئذ ان تضيفButtons وLabels وcomponents أخرى إلى الـabout-box dialog، والذى يتم عرضه فى بيئة التطوير الخاصة بالمستخدم .

بعد اختيار الخيارات المتنوعة المتاحة لإنشاء الـActiveX، يمكنك تحرير وبرمجة الـunits الناتجة فى مشروع المكتبة . عندما تنتهى، اختر Project|Build



all... لها compile الـ control . هذا يؤدى الى إنشاء ملف ocx. على القرص الذى يحتوى على الـ ActiveX .

استخدام الـ ActiveX:

بعد بناء الـ ActiveX، يجب ان تقوم بتسجيله مع الـ Windows لتجعل الـ control متاحة للتطبيقات الاخرى (بما فيهم Delphi). وهذا أمر سهل-اختر امر الـ Run|Register ActiveX Server. يجب ان يكون مشروع مكتبة الـ ActiveX مفتوحاً حتى يمكن تشغيل هذا الامر.

يسجل Delphi الـ control (أو الـ controls) فى مشروع مكتبة الـ ActiveX الحالى. عندما ينتهى، Delphi يعرض information dialog والذى يخبرك ان العملية قد تمت بنجاح. ان الـ control متوفراً الآن للاستخدام، ولكن، لم يتم تثبيته بعد على لوحة Delphi. لتفعل هذا، قم بخطوة واحدة إضافية.

Tip

فكرة: لإزالة الـ ActiveX control من النظام، افتح ملف المشروع الخاص به واختر Run|Unregister ActiveX Server. هذا يؤدى الى إزالة الـ control من سجل الـ Windows .

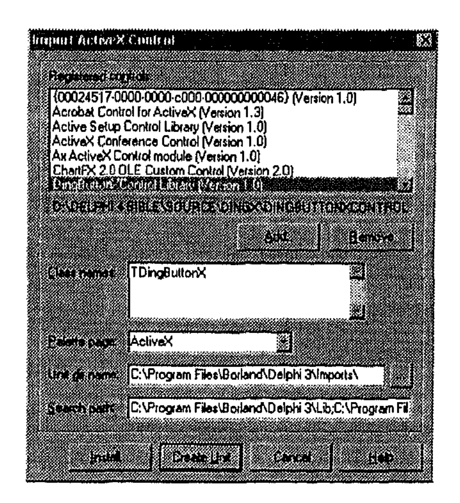
لإستخدام الـ ActiveX component فى Delphi، اختر أمر الـ Component|Import ActiveX Control. هذا يؤدى الى إظهار الـ Import ActiveX Control dialog الموضح فى شكل (٢٠-٥). ويذكر الـ dialog كل الـ ActiveX المسجلة مع الـ Windows .

اختر واحداً من الـ controls المسجلة لتدخله فى Delphi. يمكنك اختيار الـ palette page لتثبيت الـ controls، ولكن افضل ما يكون هذا فى الصفحة الافتراضية، ActiveX. اضغط زر الـ Install لإدخال الـ ActiveX فى Delphi .

Note

ملحوظة: ان الخطوات الموجودة فى هذا الفصل هى لاغراض العرض الخالصة. ولان ادخال ActiveX فى Delphi يؤدى بالفعل الى إنشاء غلاف Delphi component حول الـ ActiveX، فإنه من السخف ان يؤدى هذا العمل على الـ ActiveX الذى هو فى الحقيقة غلاف ملف حول الـ Delphi component . وفى جميع الحالات، من الافضل ان تستخدم

Delphi component الاصليـة مع Delphi. ولكن يمكنك اسـتـخـدام الخطوات المشار إليـهـا فى هذا الفصـل لإضـافة ActiveX أخرى فى Delphi للاسـتـخدام كـ component.



شكل (٢٠-٥): ادخال ActiveX فى Delphi باستخدام الـ Import ActiveX Control dialog

افكار للمستخدم الخبير

* يمكنك توفير online help للـ components الخاصة بك. عندما يـختار المستـخـدمـون component ويضـغطوا F1، يبـدأ Delphi البـحـث فى online help مفهرساً على class الكلمة الاساسية <name>_ ، حيث ان <name> هى اسـم الـ class الخـاص بك. لمزيد مـن المعلومـات انظر العنوان التـالى فى online help "دمج الـ help الخـاصـة بك فى Delphi".

* ان إزالة component من على VCL لا يؤثر على أى تطبيقات تستخدم هذا الـ component. ولكن، يجب تثـبـيـت الـ component قـبل ان تتمكن من تحميل أى مشروع يستخدمه.

* لإنشاء خاصية تكون متاحة فقط فى وقت التشغيل، ولكنها غير مرئية فى نافذة الـ Object Inspector، ضع تعريف الخاصية فى القطاع الـ public

للـ class ، ولكن لا تنشره . (فى الواقع ، لا يمكن تعريف خاصية public على إنما published ، مما قد ينشئ مسارات وصول متضاربة الى الحقل المرتبط بها . ان الخاصية إما ان تكون public ، أو تكون published ؛ فلا يمكن ابداً . ان تكون الاثنين معاً) .

* يستطيع Delphi أن يقوم بالـ compile للمشروعات فقط . لـ compile الـ unit تحت التطوير ، استخدمها فى مشروع أختبارى . يقوم Delphi بالـ compile للبيانات المعدله فى component units على حسب الحاجة عندما تقوم بتثبيتها .

* يمكن ان تكون الخصائص فى القطاعات الـ protected أو الـ published أو public للـ class . يمكنك تعريف الخصائص فى القطاع الـ private للـ class ، ولكن لن يكون هناك جدوى من ان تفعل هذا لان class unit تستطيع الوصول الكامل لكل حقول البيانات الخاصة . ان غرض الخاصية هو توفير وصولاً محكماً للبيانات التى فى قطاعى الـ protected و الـ private .

* يقوم الـ Object Inspector بصورة تلقائية بفرز اسماء الخاصية ابجدياً . يمكنك تعريف الخصائص بأى ترتيب فى الـ class . وهذا لا يؤثر على ترتيبها فى الـ Object Inspector .

* تخزن الـ VCL الخصائص بحسب ترتيب التعريف ، ولكن يحدث التحميل فى أى ترتيب تكون عليه البيانات فى ملف الـ dfm. للـ form . وهذا الترتيب يمكن ان يكون نفس ترتيب تعريف الخاصية ، ولكن يمكن ان يكون مختلفاً . على سبيل المثال ، يمكن للمستخدمين تحرير ملف الـ form كنص وتغيير قيم الخاصية وترتيبها . لا تقم ببناء components تعتمد على ترتيب تحميل الخاصية .

* يمكن ان تكون الخصائص من أى نوع بيانات فيما عدا الملفات ، ولكن هناك قيود اضافية على انواع الخصائص التى يمكن ان للـ class أن تجعلها publish . لا يمكنك نشر الـ Pascal arrays ، مثلاً ، ولا يمكنك ايضاً أن تجعل خصائص القراءة فقط أو الكتابة فقط لان الـ Object Inspector

يجب ان تقـرأها ويكتـبـهـا . ويمكـن ايضـاً للخـصـائص الـ Published الا تسـتـخـدم أنواع بيـانـات الـ Real أو الـ Extended الخـاصـة بالـ Pascal لخـصـائص الـ floating point، اسـتـخـدم Single أو Double أو Comp .

* ان packages وقت التشغيل القياسية يتم تسميتها بطريقة عشوائية نوعاً مـا . على سـبـيل المثـال، ان الـ VCL40.bpl والـ INET40bpl والـ TEEDB40.bpl تعتبر جميعاً packages وقت تشغيل والطريقة الوحيدة لمعرفة هذه الحقيقية عن الملفات هى من خلال امتدادات اسماء ملفاتها .

* ان اغلب packages وقت التصميم يتم تسميتها بطريقة متوافقة . على سـبـيل المثـال، الـ DCLSTD40.bpl والـ DCLDB40.bpl تعـد packages وقت التـصـمـيم . وللأسف، فإن هذا التـصـمـيم يكسـره الـ IBEVNT40bpl، بـالـرغـم مـن ان هـذا الـ packages فى الـ IBEventAlerter component Samples palette، وكـمثـال، فـإن اسـتـخـدامـه يكون "على مـسـئـوليـتك" . ان Borland لا يؤيد الـ components الـ sample ولقد تغيرت هذه الـ component أو اختفت فى بعض الحالات على مدى حياة Delphi .

* ان packages وقت التـشـغـيـل يتم تخـزينهـا فى الـ C:\Windows\System . وفيـمـا عـدا القليل من الـ exceptions، يتم تخزين packages وقت التـصـمـيم فى دليل الخـاص بـ Bin\ الخـاص بـ Delphi.

* من المهم ان تفـهـم الفرق بين قـائمة الـ packages المطلوبة من قبل المستند الجـديد وقـائمـة الـ units التى يجب ان يتـضـمنهـا الـ package الجـديد . فالقاعدة الأولى تشير الى الـ units الخارجية-على سبيل المثال، واحدة تحتوى على (subroutine) الذى تحتـاجه و الـ package units الجـديد . والقاعدة الثانية تشير الى الـ units التى يجب ان يعمل عليهاالـ package الجديد . ويضم المستند هذه الـ units، بنفس الطريقة تقريباً التى تضم بها الـ class بياناتها و الـ code .



* ان package وقت التصميم هو ببساطة package يذكر اسماء package وقت التشغيل الأخرى فى التعريف الخاص به .

* ان compiles الـ package يؤدى بصورة تلقائية الى compiles أى units من الـ compiles بسبب التغير فى بيانات الملف الخاص بها (أى ، القديمة) منذ آخر مرة ثم فيها الـ compile . لذلك قم دائماً بـ compile لـ package ان compile الـ units منفردة لا يؤدى الى تحديثها فى الـ package .

المشروعات التى يمكنك تجربتها

(٢٠-١) : قم بانشاء Date and Time والذى يمكنك إضافته على form أضف ميزة التنبيه الذى يدق جرساً أو يبعث رسالة . يمكنك استخدام حقل بيانات string-list لإدخال التواريخ والوقت المراد التنبيه عنها .

(٢٠-٢) : اضف خاصية الـ Font للـ TBarChart . إنشئ الـ Font للـ form font (انظر تعيين الـ Paint method للـ Canvas.Font كاشارة حول كيفية فعل هذا) .

(٢٠-٣) : قم بتحسين معالجة الـ exception الخاصة بالـ TBarChart . بدلاً من حذف نقاط البيانات اذا ما حدث exception ، جرب واحدة من الاستجابات المقترحة الاخرى . على سبيل المثال ، يمكنك عرض رمز قنبلة بدلاً من display ، أو يمكنك اختبار نقاط البيانات الفردية بشأن صلاحيتها بطريقة اخرى .

(٢٠-٤) : بإنشاء نسخة bitmap للـ RadioButton مع رمز مخصص (جوهرة ، مثلاً) فى مكان النقطة السوداء المعتادة .

(٢٠-٥) : متقدم . قم بإنشاء component أيقونة متحركة يعرض bitmaps متتالية . يمكنك استخدام الـ component لعرض صور متحركة أثناء ملف ، أو طباعة ، أو عمليات أخرى طويلة .

(٢٠-٦): متقدم. أضف خاصية array للـ TBarChart class التى تنشر array اللون الثابت الحالى للـ unit. وهذا الـ array يوفر ١٦ قيمة من قيم الألوان لرسم bars متعاقبة. (للحصول على أكثر من ١٦ bars، تتكرر الألوان). قم بتحسين الـ BarChart بجعله ممكناً للمستخدمين ليحرروا قيم الألوان هذه.

(٢٠-٧): قم بتحويل الـ BarChart الخاص بهذا الباب الى ActiveX.

ملخص:

* إن كتابة components خاصة بك ليس عملاً بسيطاً، ولكنه يمنح المبرمجين المتقدمين أدوات وإمكانات إضافية ليست متاحة لمطورى التطبيقات. يمكنك أن تؤسس الـ components الجديدة على أى من الـ components المتوفرة مع Delphi، أو يمكنك إنشاء components جديدة تماماً.

* الـ component يعتبر unit، وتعريف class خاصة مع متطلبات Delphi للـ components البصرية. لتثبيت الـ component على لوحة الـ VCL، يجب إدخاله فى package. استخدم أمر الـ ...File|New وعنصر الـ Packages لإنشاء package جديد.

* تقوم الـ Components بنشر خصائص لتوفير methods الوصول لقراءة وكتابة حقول بيانات الـ class. يستطيع المستخدمون استعراض وتحرير الخصائص الـ published والتى لها methods وصول للكتابة فى نافذة الـ Object Inspector.

* الـ Components تتطلب constructor و destructor methods الـ components البصرية بتنفيذ الـ Paint method لإنشاء مظهرها على الشاشة، فى كلاً من الـ form فى وقت التصميم، وفى البرنامج فى حالة تشغيله.

* لإزالة الأخطاء من الـ components، قم بإنشائها تحت تحكم البرنامج بدلاً من تثبيتها على لوحة الـ component هذا يجعل من الممكن استخدام

الـ debugger الخاص بـ Delphi بتخطى برمجة الـ component بخطوة واحدة لإدخال breakpoints ومتغيرات فحص .

* يمكن أن تكون الخصائص أى نوع من أنواع البيانات فيما عدا الملفات والـ Pascal arrays . يمكنك إنشاء خاصية array تقوم باستدعاء methods وصول للقراءة والكتابة ، ولكن هذه الخصائص يتم استخدامها كـ array . استخدم هذه التقنية لإنشاء هياكل بيانات array مرتبط وموزع .

* يمكنك استخدام الـ ActiveX Control Wizard لتحويل أى Delphi component الى ActiveX يمكنك أيضاً إدخال أى ActiveX فى Delphi أو استخدامه كـ component . ولكن ، من الأفضل دائماً أن تستخدم Delphi components الأصلية مع Delphi.

أتمنى أن تجد شيئاً ذا قيمة عالية فى الباب القادم ، والذى يجمع أفكاراً ، وحيلاً ، وتقنيات لشحذ مهاراتك فى برمجة Delphi .

*****



الباب الحادى والعشرين

تطوير مهارات Delphi الخاصة بك

محتويات هذا الباب:

- Components
- ادوات الـCommand-line
- تطبيقات الـ CRT
- functions نافعة
- معلومات نوع وقت التشغيل
- Online help
- Dynamic Link Libraries
- event handlers للرسالة والتطبيق
- File streams
- تطبيقات الـ Internet
- مزيداً من الافكار

بينما كنت اكتب هذا الكتاب، جمعت العديد من الافكار والحيل والهامة لأقدمها فى هذا الباب الآخير. ان هذا الباب ليس له ترتيب أو هدف معين، ولكن من المؤكد أنك ستجد العديد من الملحوظات والاقتراحات التى تساعدك على شحذ مهارتك فى برمجة Delphi. فى هذا الفصل سوف نتعلم كيفية استخدام ادوات (command-line) النافعة، وأفكاراً حول إنشاء واستخدام online help ،

وكيفية إنشاء واستخدام الـ dynamic link libraries ، وكيفية إدخال forms وcomponents اخرى فى الـ DLL ، وكيفية استخدام معلومات نوع وقت التشغيل . يمكنك ايضاً ان تجد معلومات حول الـevent handlers الخاصة بالتطبيق ، سير الملف ، التطبيقات الـ multithreaded ، مقتطفات عن برمجة الـ Pascal مثل method overloading ، الـ arrays الديناميكية ، والـ parameters الافتراضية .

ادوات (الـCommand-Line):

يأتى Delphi بأدوات الـ command-line التى يمكنك استخدامها لإنشاء تطبيقات الـ Windows . ان النتائج هى نفسها التى تحصل عليها مع البيئة المتكاملة ، ولكن ادوات الـ command-line تسمح بالتطوير باستخدام محررين ثانويين بدلاً من بيئة Delphi . يمكنك ايضاً اختيار خيارات complier لا يفهمها الا الخاصة وإصدار ملفات خريطة رابطة للاستخدام اثناء إزالة الاخطاء . وتعتبر ادوات الـ command-line قيمة ايضاً فى إنشاء برامج اختبارية قصيرة ولتعليم تقنيات برمجة الـ Pascal من الكتب .

على القرص المدمج: ان الخطوة الأولى فى استخدام ادوات الـ command-line الخاصة بـ Delphi هى الوصول الى (command-line prompt) . يمكنك عندئذ إما ان تترك الـ Windows (استخدم أمر الـ Shut Down لزر الـ start واختر "Restart in MS-DOS mode." . أو ، اذا لم تكن تريد ان تترك الـ Windows ، افتح نافذة الـ DOS باستخدام الـ Start|Programs . إدخل path لتتأكد مما اذا كان دليل الـ\bin الخاص بـ Delphi موجود على مسار النظام . اذا لم يكن موجوداً ، قم بتشغيل ملف المجموعة Delpath.bat فى القائمة (٢١-١) . يوجد هذا الملف فى دليل الـSource\Misc على القرص المدمج . قد يكون عليك تعديل string المسار إعتماداً على نسختك من Delphi واين ثبتها .

القائمة (٢١-١):Misc\Delpath.bat

```
@echo off
rem
```



```
rem delpath.bat-Configure PATH for Delphi command line
rem
set path=%path%;"C:\Program Files\Borland\Delphi 4\bin"
echo Configured for Delphi command line
path
```

يمكنك الآن compile تطبيقات الـ CRT المستقلة، وإكمال مشروعات الـ Delphi لـ Windows . على سبيل المثال، مع وجود دليل الـbin\ على المسار، استخدم نافذة الـ DOS للإنتقال الى أى دليل به مشروع لـ Delphi . لتجربة هذا، إنتقل الى نسخة من تطبيق الـPolyflow الخاص بهذا الكتاب. ومن الـ DOS prompt، إدخل أمراً مثل هذا :

```
cd "\Delphi 4 Bible\Source\polyflow"
```

لاحظ علامات التنصيص، والمطلوبة للادلة ذات الكلمات المتعددة ذات الفراغات مثل Delphi 4 Bible لـ compile التطبيق، قم بتشغيل الـ dcc32 (compile خط امر Delphi ذى الـ ٣٢بت). من الـDOS prompt. هذا يؤدى الى تشغيل الـBorland Pascal compiler. ويبحث الـ compiler عن ملفات الـ pas . حسب النظام الافتراضى، لذلك، لكى تقوم بعملية الـ compile لتطبيق الـ Windows، يجب ان تدخل امتداد اسم الملف dpr.. ادخل النص التالى بخط سميك. يجب ان تشبه شاشتك السطور الموضحة هنا، بالرغم من ان ارقام النسخ وتواريخ حق الطبع قد تختلف :

```
D:\Delphi 4 Bible\Source\Polyflow>dcc32 polyflow.dpr
Borland Delphi for Version 12.0 Copyright (c) 1983,98
Inprise Corporation
Polyflow.dpr(14)
15 lines, 0.75 seconds, 149128 bytes code, 4397 bytes data.
```

بفرض أنك لم تتلقى أى اخطاء compiler، يمكنك الآن كتابة Polyflow واضغط Enter لتشغيل البرنامج استخدم هذا الـ method لـ compile مشروعات والتى تريد ان تختارها مع خيارات بواسطة command-line compiler و linker للحصول على قائمة بالخيارات المتاحة، اكتب dcc32 واضغط Enter . على سبيل المثال، لـ compile برنامج وتعريف رمز شرطى، يمكنك إدخال أمر مثل هذا :

dcc32 -DDEBUGMODE polyflow.dpr

تحذير: اذا حاولت compile تطبيقاً قديماً تم كتابته باستخدام Delphi، قد تتلقى خطأ بأن ملف الـ resource ذى الـ ١٦ بت الخاص بالمشروع لا يتماشى مع linker ذى الـ ٣٢ بت الخاص بـ Delphi. وقد يحدث هذا حتى بعد ان تقوم بتحديث وcompile التطبيق بنجاح باستخدام البيئة المتكاملة لـ Delphi. والطريقة الوحيدة للخروج من هذه المشكلة هى ان تنشئ تطبيقاً جديداً ثم تضع ملفات الـ source القديمة فى المشروع.

تطبيقات الـ CRT:

ان النسخ الأولى من Delphi قد وفرت تطبيقات الـ CRT لكتابة فقط البرامج الاختبارية. وكان هذا ضرورياً لان الـ Windows 3.1 ذى الـ ١٦ بت لم يكن يؤيد تطبيقات الـ console، كما يفعل الـ Windows 95 والـ Windows 98 والـ Windows NT. ان قالب تطبيق الـ Crt لم يعد متاحاً، ولا فى الـ WinCrt التى قدمت مخرجات تشبه الـ DOS فى نافذة جرافيكية. ولكن، مع النسخ الاحدث لـ Delphi، يمكنك استخدام الخطوات التالية لإنشاء تطبيق CRT المستقل والذى يتم تشغيله من الـ DOS prompt.

ان تطبيق الـ CRT هو ببساطة برنامج نص فقط ولا يتطلب from ولا يستخدم components. ويعتبر تطبيق الـ CRT هو اسلوب الاقتصاد-وهو لا يكلفك الكثير من الوقت والمجهود، ولكنه يذهب بك حيث تشاء بسرعة.

إنشاء تطبيق الـ CRT:

استخدام تطبيقات الـ CRT لاختيار methods برمجة الـ Object Pascal، ولا ختبار الـ algorithms، ولتشغيل نماذج النص العادى من البرامج التعليمية للـ Pascal. لإنشاء تطبيق CRT، اقترح الخطوات التالية. قد يكون هناك طرق اخرى لتحقيق نفس النتائج النهائية، ولكننى اجد هذا الـ method هو الأسهل:

١- راجع الفصل السابق لتحدد الـ PATH الخاص بالنظام ليشمل دليل الـ bin\ الخاص بـ Delphi. بالرغم من انه من الممكن compile تطبيقات الـ CRT



من داخل البيئة المتكاملة لـ Delphi ، الا ان compile تطبيقات الـ CRT المستقلة يعتبر اسهل من الـ DOS prompt . افتح نافذة الـ DOS prompt الآن وقم بتشغيل ملف مجموعة الـDelpath المذكور فى الفصل السابق .

الملف موجود فى الدليل (source\misc)

٢- ارجع الى Delphi. اغلق كل الملفات الموجودة، اختر ....File|New اضغط الـ New page tab فى الـ New Items dialog، واختر الـ Text template . هذا يؤدى الى إنشاء ملف جديد، هو File1.txt، فى محرر الـcode .

٣- لإخبار Delphi ان الملف الجديد يحتوى على عبارات Pascal، احفظه فوراً مع امتداد اسم الملف pas. . هذا يمكن من ابراز الـ syntax-highlighting ، ويجعل الـ Code Insights متاحة (راجع الباب الأول) .

٤- إدخل برنامجك واحفظه . يمكنك استخدام أى برنامج Pascal، ولكن اذا لم يكن لديك واحداً، إنسخ النص من القائمة (٢١-٢) . يوجد هذا الملف على القرص المدمج فى دليل Source\Keytest .

٥- ارجع الى نافذة الـDOS prompt. إدخل dcc32 -cc keytest لـ compile برنامجك-لايجب عليك كتابة امتداد إسم الملف pas. ، ولكن ان تفعل فلا تضرر من هذا .

تحذير: عند compile تطبيقات الـ CRT المستقلة، لا تنسى ان تحدد خيار الـ cc-. هذا يخبر الـ compile ان يقوم بتشغيل مخرجات ومدخلات الـ command-line بدلاً من استخدام الـ Windows functions.

القائمة (٢١-٢): Keytest\Keytest.pas. تطبيق CRT كمثال يمكنك استخدامه لاختبار لوحة المفاتيح وتجربة ادوات الـ command-line الخاصة بـ Delphi. اضغط Ctrl+C للإنهاء

```
program KeyTest;
var
 Ch: Char;
```



```
begin
 Writeln('Keyboard tester. Press +Enter.');
    Writeln('To quit, press Ctrl+C,');
    Writeln('or close the window.');
    Writeln; { Output a blank line }
    repeat
  Read(Ch);       { Read from keyboard }
       Writeln(Ord(Ch):4);  { Show its value }
      until False; { That is, "forever." }
end.
```

Note

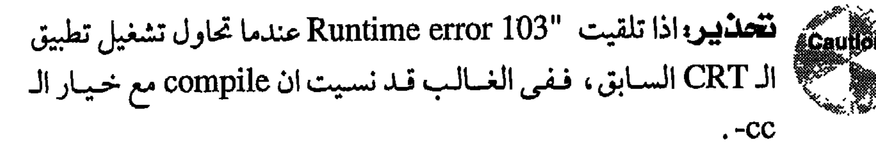
ملاحظة: من الممكن توليد console application باستخدام الـ ProjectlOptions... ، اختيار Linker ، وتشغيل الـ Console Application check box . وهذه تعتبر ايضاً حيلة جيدة لتشغيل عبارات إزالة الاخطاء Writeln التى تعرض قيم أو تشير الى عمليات برنامج متنوعة اثناء التطوير. ولكن تشير الى عمليات برنامج متنوعة اثناء التطوير. ولكن من الاسهل غالباً compile وتشغيل تطبيقات CRT الخاصة من الـ prompt command-line.

اذا اكتشفت ان تشغيل تطبيق الـ Delphi Windows يفتح نافذة DOS ، فإن هذا الخيار يكون فى الغالب قد تم تشغيله . قم بابطال اختياره ، وأعد الـcompile لإصلاح المشكلة.

اذا كان لديك بالفعل Pascal source code ، فيمكنك بالطبع استخدام محرر Delphi's code لاستعراض وتعديل عبارات البرنامج. استخدم الـ FilelOpen... لفتح الملف (تأكد من ان الـ Files من النوع المحدد pas.* كواحد من امتدادات اسم الملف المعروفة). بعد حفظ تغييراتك استخدم command-line compiler كما هو موضح لـ compile البرنامج ، والذى يمكنك تشغيله من الـ DOS prompt. واذا اردت ، يمكنك استخدام محرر نص آخر لاستعراض ، إنشاء ، وتحرير ملفات الـ source code للـ CRT-يوجد العديد من الانواع الجيدة فى السوق.

استخدم الـ Write، Writeln ، Read ، وReadln للادخال والإخراج فى تطبيقات الـ CRT على سبيل المثال، عرف متغير string، وجرب هذا البرنامج لتحفز المستخدمين على الإدخال ثم اعرض الاستجابة (لايوجد هذا النص على القرص المدمج):

```
program Yourname;
var
 S: String;
begin
 Write('Hi! What''s your name? ');
    Readln(S);
    Writeln('Your name is ', S);
end.
```



تحذير: اذا تلقيت "Runtime error 103" عندما تحاول تشغيل تطبيق الـ CRT السابق، ففى الغالب قد نسيت ان compile مع خيار الـ -cc.

function قوية:

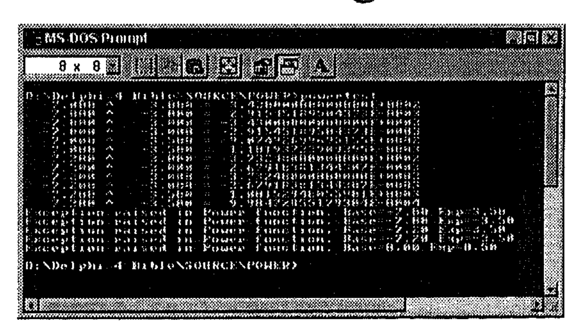
بالرغم من ان الـ Object.Pascal Math unit الخاصة لـ Delphi توفر الـ Power function التى يكون فيها قيمة الأس ، ويقدم هذا الفصل Power function التى تحدث exception لهذه العملية والعمليات الأخرى للقوة الغير المسموح به. لكى تستخدم هذه الـ function فى الـ code، انسخ نوع الـ exception class الثابت، وfunction من القائمة الموجودة بعد الفصل الى أى module، واستدع Power كما يوضح الـ Test procedure لهذا البرنامج.

لقد نشرت نسخ من الـ Power فى اماكن اخرى، ولكننى ضمنت نسخة جديدة منها هنا لإظهار كيف تتمكن الـ function الرياضية من استخدام معالجة الـ exception لتعريف الاخطاء. على سبيل المثال، من الغير مسموح به ان ترفع اساس سالباً الى اس كسرى أو ترفع الصفر الى أس غير صفرى اقل من واحد. ان

الـ Power function هنا حدث EPower exception لهاتين الحالتين وغيرها من الحالات المشابهة. قم بتشغيل البرنامج لإظهار كيف يتعامل الـ Test procedure مع هذه الانواع من الاخطاء.

On the CD-ROM **على القرص المدمج:** للقيام بهذا، افتح نافذة محفز command-line، واذا لم تكن قمت بهذا حتى الآن، قم بتشغيل ملف مجموعة الـDelpath.bat كما تم التوضيح سابقاً فى هذا الباب. إنتقل الى نسخة من دليل الـ Source\Power على القرص المدمج، وإدخل dcc32 -cc powertest لـ compile البرنامج. إدخل powertest لتشغيل البرنامج. يوضح شكل (٢١-١) مخرجات نص البرنامج التى تراها على الشاشة. توضح القائمة (٢١-٣) الـ .source code للبرنامج. قم بـ Compile وتشغيل هذا البرنامج الاختبارى من الـ DOS prompt.

On the CD-ROM **على القرص المدمج:** يمكنك تشغيل تطبيقات الـ CRT باستخدام الـ Windows Explorer، ولكن هذا يؤدى الى إغلاق نافذة الـ DOS بمجرد إنتهاء التطبيق. لمنع هذه المشكلة، افتح نافذة DOS prompt باستخدام امر الـ Programs لقائمة الـ Start. إنتقل الى دليل الـ Source\Power واكتب powertest لتشغيل البرنامج الاختبارى.



شكل (٢١-١): تشغيل برنامج الـ PowerTest يعرض هذه المخرجات فى نافذة DOS prompt

القائمة (٢١-٣): Power\PowerTest.pas. هذا البرنامج يحتوى على الـ function Power، التى يمكن ان تكون الأس من نوع الـ floating point

```
program PowerTest;
```

```
uses SysUtils;

const
 sFmt = 'Exception raised in Power function. Base=%f
   Exp=%f';

type
{ Declare exception class }
 EPower = class(EMathError}

{ Return Base raised to Exponent }
function Power(Base, Exponent: Double): Double;
 function F(B, E: Double): Double;
    begin
     Result := Exp(E * Ln(B));
        end;
begin
 if Base = 0.0 then
      if Exponent = 0.0 then
           Result := 1.0
              else if Exponent < 1.0 then
  raise EPower.CreateFmt(sFmt, [Base, Exponent])
      else
  Result := 0.0
     else if Base > 0.0 then
 Result := F(Base, Exponent)
  else if Frac(Exponent) = 0.0 then
 if Odd(Trunc(Exponent)) then
      Result := -F(-Base, Exponent)
         else
      Result := F(-Base, Exponent )
        else raise EPower.CreateFmt(sFmt, [Base,
   Exponent]);
end; { Power }
```

```
{ Test procedure }
procedure Test(Base, Exponent: Double);
begin
 try
   Writeln(Base:8:3, ' ^ ', Exponent:8:3, ' = ',
     Power(Base, Exponent));
       except
  on E: EPower do
     Writeln(E.Message);
       end;
end;

{ The following is the CRT application's main body. It
 merely calls the test procedure with various values. The
    final four values intentionally test the Power function's
    exceptions. }
begin
 test( 7,  3 );
    test( 7,  -3 );
    test(-7,  3 );
    test(-7,  -3 );
    test( 7,  3.5);
    test( 7,  -3.5);
    test( 7.2, 3 );
    test( 7.2, -3 );
    test(-7.2, 3 );
    test(-7.2, -3 );
    test( 7.2, 3.5);
    test( 7.2, -3.5);
{ These four tests produce *expected* exceptions }
 test(-7,  3.5);
    test(-7,  -3.5);
    test(-7.2, 3.5);
    test(-7.2, -3.5);
    test( 0,  0.5);
end.
```

ان الـ Power function تحــدث exception من الـ EPower class، المشـتـقـة من الـ EMathError class فى الـSysUtils unit لـ Delphi . و الـ EMathError class بدورها مشتقة من الـ class الاساس Exception . (انظر الباب التاسع عشر لمعرفة مزيد من المعلومات حول البرمجة مع الـ exception) .

لاســتـخـدام الـ Power function، يجب ان توفـر وصــولا الى الـ exception classes الخاصة بـ Delphi . لكى تفعل هذا ، اضف SysUtils unit الى امر الـuses الخاص ببرنامجك (يستطيع Delphi ان يقوم بهذا بصورة تلقائية تدمج الـ Power function فى تطبيق الـ Windows) . انسخ الـ code الموجودة فى القائمة (٢١-٥) برنامجك ، ناقص تعريف الـ program والـ uses ، والـ end النهائية .

بالإضافة الى توفير function رياضية نافعة ، تظهر الـ Power ايضاً طريقة جيدة لإنشاء exceptions رياضية توضح قيم خاطئة . على سبيل المثال ، لاحداث exceptions ، تنفذ الـ Power هذه العبارة :

```
raise EPower.CreateFmt(sFmt, [Base, Exponent])
```

ان الـ CreateFmt function، الموروثة من الـ Exception ، تأخـــذ arguments : string اوامر ، ومجموعة من القيم المحتجزة فى اقواس مربعة . عندما يحدث exception ، يحتوى حقل الـMessage الخاص بالـ EPower على string يوضح القيم التى سببت الخطأ . على سبيل المثال ، يعرض البرنامج الاختبارى رسالة خطأ لـ exception مثل :

```
Exception raised in Power function. Base=-7.20 Exp=3.50
```

بعض الـ Functions النافعة:

ان كل مبرمج لديه مكتبة من (subroutines) التى اثبتت قيمتها بمرور السنين . فيما يلى بعض الإضافات النافعة التى يمكنك ان تضعها بمكتبتك .

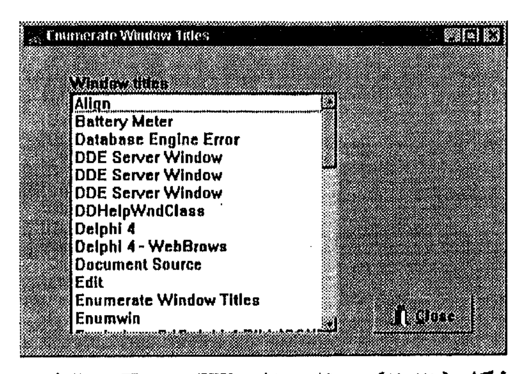
الـ Callback functions:

ان الـ callback function تعتبر (subroutines) فى برنامجك يقوم الـ Windows باستدعاءها . والاستخدام الامثل للـ callback function هو سرد

قـائمـة من العناصـر مـثل العناوين والـ handles لكل النوافذ ، أو اسـمـاء الـ fonts المتاحة . والامثلة التالية توضح التقنيتين .

On the CD-ROM

على القـرص المدمج: توجـد القـائمـة (٢١-٤) فى ملف Main.pas لمشـــــروع الـ EnumWin عـلى الـقـــــرص المدمـج فى دلـيل الـ Source\EnumWin . قم بتـحـمـيل ملف مـشـروع البـرنامج فى Delphi، ثم اضـغط F9 لـ compile والتشـغيل . كـمـا يوضح شكل (٢١-٢)، يذكر البرنامج كل النوافذ النشطة حالياً (أى، المعروفة لـ Windows) . وبالطبع ان عناوين النوافذ التى تراها تختلف غالباً عن التى تراها هنا . بالإضـافة الى توضيح كيفية ايجاد عناوين النافذة، يوضح البرنامج الطريقة الصحيحة للوصول الى كل النوافذ، وهو ما قد تفعله لعدة اسباب- على سبيل المثال، لتحدد ما اذا كان التطبيق الحـالى فى حـالة تشـغيل بالفـعل، لإرسـال رسـالة الى كل النوافـذ، أو لـتـمـكن المستخدمين من التحول الى نافذة معينة .



شكل (٢١-٢): برنامج الـ EnumWin يستخدم الـ callback function لذكر كل النوافذ "الحيلة"

القائمة (٢١-٤): EnumWin\Main.pas

```
unit Main;

interface

uses
```

```
      Windows, SysUtils, Classes, Graphics, Controls,
      Forms, Dialogs, StdCtrls, Buttons;

type
 TMainForm = class(TForm)
    ListBox1: TListBox;
       Label1: TLabel;
       BitBtn1: TBitBtn;
       procedure FormCreate(Sender: TObject);
      private
    { Private declarations }
      public
    { Public declarations }
      end;

var
   (MainForm: TMainForm;

implementation

{$R *.DFM}

{ Windows calls this function, passing in Handle a
reference
 to each alive window. Param is not used in this example. }
function EnumWinProc(Handle: HWnd; Param: Longint):
Boolean;
 Stdcall; { Use this, not export, in Windows 95, 98 & NT }
var
 Sz: array[0 .. 132] of Char; {Holds result of
GetWindowText}
begin
 Result := True; { Always successful }
   { Call Windows to obtain each window's caption, and
   then
    add the returned string to the form's ListBox. }
      if GetWindowText(Handle, Sz, Sizeof(Sz)) 0 then
```

```
      MainForm.ListBox1.Items.Add(StrPas(Sz));
end;

{ Enumerate all alive windows by passing to Windows the
 address of the preceding callback function. }
procedure TMainForm.FormCreate(Sender: TObject);
begin
 EnumWindows(@EnumWinProc, 0); { 0 is an unused
parameter }
end;

end.
```

يستخدم البرنامج ListBox object واحد لحمل قائمة عناوين النوافذ. لملء الـ ListBox object بهذه المعلومة، قمت بإنشاء الـ OnCreate لـ form التطبيق. ينفذ الـ FormCreate procedure عبارة واحدة هي:

```
EnumWindows(@EnumWinProc, 0); { 0 is an unused
parameter }
```

الـ EnumWindows يعتبر Windows procedure يتلقى arguments عنوان الـ callback function، و parameter اختياري (أو Longint) - أي، object ذي ٣٢ بت على انه pointer. وهذا المثال لا يستخدم الـ argument الثانية، لذا حددتها بصفر. وقد يستخدمها برنامج آخر لتمرير العنوان الخاص بـ object البيانات الى الـ callback function .

ان الـ Callback function يجب ان يتم تعريفها بطريقة صحيحة وهذا أمر غاية فى الأهمية لان الـ compiler لا يتأكد من صحة syntax . عليك ان تستشير بحرص كل المصادر الممكنه للتوثيق لتحدد الـ syntax الصحيح لـ callback function التى تريد استدعاءه (ان الـ Windows.pas فى دليل الـ Source\Rtl\Win الخاص بـ Delphi's يعتبر مفيد فى هذه الحالة). وفى البرنامج، يتم تعريف الـ callback function كما يلى:

```
function EnumWinProc(Handle: HWnd; Param: Longint):
Boolean;
 Stdcall;
```

Note **ملحوظة:** فى الـ Windows ذى الـ ١٦ بت، يتم تعريف هذه الـ function بطريقة مختلفة، باستخدام الـ Export designator. فى الـ Windows 95 والـ Windows NT ذى الـ ٣٢ بت، يجب تغيير الـ Export الى Stdcall. هذا يضمن تمرير الـparameters فى ترتيبها الصحيح.

ويتلقى الـ EnumWinProc إثنين من الـ parameters : الـ Handle الذى يشير الى نافذة واحدة، والـ Param والذى، كما ذكرت، لا يتم استخدامه فى هذه الحالة. تقوم الـfunction بإعادة نتيجة True أو False. يدخل الـ الـ EnumWinProc النتيجة True للأستمرار فى تكرار كل النوافذ الحية حتى تنتهى، ولكن يمكن ان تدخل False لإيقاف الاستمرارية.

تذكر أن الـ EnumWinProc ليست Windows function. إنها function فى برنامجك يستدعيها الـ Windows. ولهذا السبب، ان الـ Windows يهتم فقط بمكان (عنوان) الـ function- فإسمها لا يهم. اذا اردت ان تعيد تسمية الـ EnumWinProc فهذا أمر يخصك. فالـ Windows لا يهتم. لقد ذكرت هذا لأنه فى العديد من القوائم الـpublished ، تجد الـ Windows يستدعى function ذات هجاء مختلف.

ان الـ code الموجودة داخل الـ callback function تحدد الـ Result الخاص بالـ function بـ True، ثم تستدعى الـ GetWindowText function الخاصة بالـ Windows باستخدام عبارة الـ if التالية :

```
if GetWindowText(Handle, Sz, Sizeof(Sz)) 0 then
  MainForm.ListBox1.Items.Add(StrPas(Sz));
```

يقوم هذا الـ code بتمرير الـ GetWindowText الى الـ Handle الخاص بالنافذة و arguments إضافيتين : مكان لتخزين caption الخاص بالنافذة، وحجم هذا المتغير. وللتبسيط، قمت بإنشاء Char array ذى ١٣٣ بايت، وهو كبير بدرجة تكفى لحمل caption ذو ١٣٢ حرفاً. هذا يعتبر كافياً لأى عنوان نافذة. ان الـ Sizeof function تمرر الحجم بالبايت للـ GetWindowText التى تم برمجتها على الا تخزن المعلومات فى string object اكثر من المحدد لها.

اذا قـامـت الـ GetWindowText function بارجـاعTrue، إذن يوجـد عنوان النافذة فى المتغير Sz ، والذى تم إضافته الى الـ ListBox1 الخاص بالـform . لاحظ ان الـMainForm قـد تم الإشـارة إليـهـا لتـعـريـف الـListBox1 object . وهذا يعـتـبـر ضـروريـاً لان الـ callback function ليـسـت عـضـواً فى الـ TMainForm class للتطبيق .

: Procedure instances

ان الـWindows ذى الـ ١٦ بت كان يتطلب مجهوداً خرافياً لإستدعاء الـ procedures والـ functions فى code التطبيق الخاص بك . فى الماضى ، كان من الضرورى إنشاء procedure instance، والمعروفة ايضاً بـ thunk، لاستخدام الـ callback function المستقلة . ولكن لم يعد هذا ضـرورياً الآن مع الـWindows 95 و الـ98 Windows والـWindows NT ، والذى يسـتـدعى الـ functions التطبيق مباشرة .

على القرص المدمج: لعرض كلتا التقنيتين ، توضح القائة (٢١-٥) الـ source code الرئيسية لمشروع الـ Enumfon . على القرص المدمج ، يمكنك ان تجـد ملفـات البـرنامج فى دليل الـ Source\ Enumfon . وبتشغيل البرنامج تحصل على قوائم بكل اسماء الـfont المتاحة-يمكنك استخدام برمجة مشابهة لملئ مربع القائمة الذى يختار منه المستخدمون الـ font . لقد قمت بتحديث هذه التقنيات الجديدة ، فبدلاً من إعادة إنشاء الـ code ، قمت بإدخال conditional compilation directives . بهـذه الطريقة ، يستطيع البرنامج الـ compilation مع كل نسخ Delphi والـ Windows . اذا اردت تحديث تطبيقات قديمة ، يمكنك استخدام تقنيات مشابهة للحفاظ على توافق الـ ١٦ بت يوضح شكل (٢١-٣) عرض البرنامج .

القائمة (٢١-٥): Enumfon\Main.pas. يوضح البرنامج كيفية سرد اسماء الـ font باستخدام النسخ ذات الـ ١٦ والـ ٣٢ بت للـDelphi والـWindows

```
unit Main;
interface
uses
```

```
    Windows, SysUtils, Classes, Graphics, Controls,
    Forms, Dialogs, StdCtrls, Buttons;

type
 TMainForm = class(TForm)
    ListBox1: TListBox;
      Label1: TLabel;
      BitBtn1: TBitBtn;
      procedure FormCreate(Sender: TObject);
     private
    { Private declarations }
     public
    { Public declarations }
     end;

var
 MainForm: TMainForm;

implementation

{$R *.DFM}

{ ======================================== }

{$IFDEF VER80} // Delphi 1.x, 16-bit Windows only

function EnumFontsProc(var LogFont: TLogFont;
 var TextMetric: TTextMetric; FontType: Integer;
   Data: TListBox): Integer; export;
begin
 with TStrings(Data) do
    Add(StrPas(LogFont.lfFaceName));
      Result := 1; { Continue enumeration until done }
end;

procedure TMainForm.FormCreate(Sender: TObject);
var
 Proc: TFarProc;
begin
 Proc := MakeProcInstance(@EnumFontsProc, HInstance);
```

```
   try
 EnumFonts(Canvas.Handle, nil,Proc,Pointer
(ListBox1.Items));
 finally
 FreeProcInstance(Proc);
      end;
end;

{ ================================== }

{$ELSE} // Delphi 2 and higher, 32-bit Windows 95, 98, and NT

function EnumFontsProc(var LogFont: TLogFont;
 var TextMetric: TTextMetric; FontType: Integer;
   Data: Pointer): Integer; stdcall;
var
 TheList: TStrings;
begin
 TheList := TStrings(Data); // Get StringList passed in Data
   TheList.Add(LogFont.lfFaceName); // Add font's name
   Result := 1;      // Continue until done
end;

procedure TMainForm.FormCreate(Sender: TObject);
var
 DC: HDC;
begin
 DC := GetDC(0); // Get a device context handle
   try
    EnumFonts(DC, nil, @EnumFontsProc,
        Pointer(ListBox1.Items)); // Pass data to callback fn
          finally
      ReleaseDC(0, DC); // Release system resource
      end;
end;
```

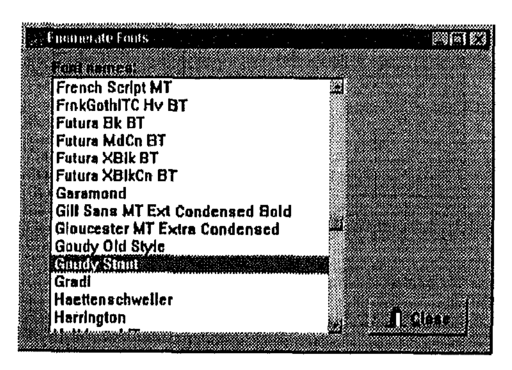
```
{$ENDIF}

{ ============================================ }

(*
{ The next procedure is *far* simpler than either of
the preceding methods, and it also works with all versions
   of Delphi and Windows! Remember: Don't work harder
  than
   you have to; rather than call Windows subroutines and
  use
   callback functions, look for a Delphi component that
   does what you want. You can probably find one! }

procedure TMainForm.FormCreate(Sender: TObject);
begin
 Listbox1.Items := Screen.Fonts;
end;
*)

end.
```



شكل (٢١-٣): تطبيق الـ Enumfon يذكر كل اسماء الـ font المتاحة

الحصول على معلومة اسم الـ font:

قبل قراءة شرح الـ FormCreate المستخدمة فى الـ Enumfon، ارجع الـ procedure البسيط الموجودة فى آخر القائمة داخل اقواس تعليق. ان الـ FormCreate procedure يوضح ابسط method للحصول على معلومة اسم الـ font قم بتعيين الـ Screen.Fonts لـ TStringList object مثل خاصية الـ Items فى ListBox. وهذه العبارة:

```
Listbox1.Items := Screen.Fonts;
```

توضح واحداً من مبادئ البرمجة الـ Delphi استخدم دائماً objects و component methods اينما كان ممكناً. لا تعقد الامور باستدعاء Windows API subroutines واستخدام الـ callback functions بلاداعي.

والسبب الآخر لاستخدام objects هو ان الـ VCL palette الخاصة بـ Delphi تضم code توضح بعض غرائب الـ Windows ذي الـ ٣٢ بت. على سبيل المثال، ان الـ EnumFonts function القياسة للـ Windows لا تدخل دائماً كل اسماء الـ font مثل اليابان. وتتغلب الـ VCL TScreen palette على هذه المشكلة باستدعاء الـ EnumFontFamilies لتقديم قائمة اكثر شمولاً عما هي موجودة في الـ EnumFonts. ان الـ VCL الخاصة بـ Delphi تحتوى على العديد من الاشياء الأخرى التى يمكنك الانتفاع بها باستخدام objects بدلاً من استدعاء الـ Windows API functions.

توضح الـ Enumfon ثلاث تقنيات لذكر اسماء الـ font، إثنين منهم يستخدماً الـ callback functions الخاصة بالـ Windows. والـ function الموضحة الأولى تكون لتطبيقات الـ ١٦ بت للـ Windows 3.1 ؛ والثانية لتطبيقات الـ ٣٢ بت للـ Windows 95, 98, & NT .

اذا كان يجب عليك استخدام الـ callback functions ، فإليك طريقتين لتعقيد code برنامجك وبالطبع ، هناك اسباب لإنشاءالـ callback functions مثل enumerating النوافذ واستخدم تقنيات الـ Windows الغريبة الغير متوفرة بواسطة Delphi components . ولكن اريد ان اذكر ان التقنيات التالية يجب ان تحفظ لاوقات لا يوجد فيها بدائل اخرى .

فى الـ Windows ذى الـ ١٦ بت ، إنك تعرف الـ callback function مثل EnumFontsProc باستخدام الكلمة الاساسية export . لاستدعاء الـ function، من الضرورى اولاً ان تنشيء حالة procedure instance تعمل موصل للاستدعاءات من الـ Windows عائدة الى code التطبيق . لكى تفعل هذا ، يستدعى البرنامج الـ MakeProcInstance لـ function معينة ، ثم يمرر النتيجة الى دالة الـ Windows التى تريدها . ها هو حد التمييز الاساسى :

```
Proc := MakeProcInstance(@EnumFontsProc, HInstance);
```

```
try
 EnumFonts(Canvas.Handle, nil, Proc,
    Pointer(ListBox1.Items));
finally
 FreeProcInstance(Proc);
end;
```

أولاً، يتم استدعاء الـ MakeProcInstance لإنشاء procedure instance من نوع الـTFarProc للـ callback function، EnumFontsProc في هذا المثال. الـ procedure instance هذه يتم تمريرها الى الـ EnumFonts مع device context handle، و pointer للـ TStringList. وبالطبع، ان الـ Windows لا يعرف شيئاً عن الـ TStringList- الـ parameter الاخير، والذى يكون إختيارياً والذى يمرر إلى الـ call back Function الخاص بالبرنامج. من الضرورى استخدام الـ try-finally block، كما هو موضح هنا، لـ procedure instance بحيث يتم تحرير بغض النظر عما اذا حدثت ايه exceptions اثناء استدعاءالـ EnumFonts. ولك ان تتوقع حدوث مشاكل ضخمة اذا لم تقم بمسح الـ Windows resources بهذه لطريقة.

ومن على السطح، تبدو تقنية الـ ٣٢ بت المقابلة بسيطة، ولكنها فى الواقع تختلف عن الـ method السابق القديم. ان الـ callback function، وهى EnumFontsProc، يتم تعريفها بطريقة مشابهة، بالرغم من اننى قد اوضحت طريقة افضل لتمرير الـ Data parameter على انه Pointer. بمعنى آخر، الـ Data تشير الى أى بيانات يتم تمريرها الى الـ callback function. ونتيجة الـ function تكون واحدة، وهى Integer. ولكن لاحظ ان الـ sStdcall يتم استخدامها بدلاً من الـ export. وهذا امر هام. تأكد من ان الـ callback function الخاصة بك تستخدم دائماً الـ sStdcall.

وداخل الـ callback function، يتم بدء متغير مؤقت، هو الـ TheList، باستخدام الـ Data pointer الذى تم تمرير على انه parameter. هذا لا يؤدى الى نسخ الـStringList، انه يجعل الـ TheList يشير الى الـ object كمشير اكثر سهولة فى العبارات اللاحقة. على سبيل المثال، لإضافة كل اسم font، يستدعى

البرنامج الـ TStringList.Add method . لقد قدم الـ Windows بالفعل اسم الـ font فى الـ LogFont parameter ، وهذا هو كل ماهو مطلوب لإضافة كل اسم الى الـ ListBox الخاص بالبرنامج . وكما سبق ، ان الناتج لواحد يتم تمريره للخلف لاكمال العملية حتى يمرر الـ Windows جميع الـ fonts .

ان OnCreate الخاص بالبرنامج للـ form الرئيسية يوضح كيف يستعد البرنامج ذو الـ ٣٢ بت لاستخدام الـ callback function . كان يمكننى ان استعير الـ Canvas.Handle كـ device context ، ولكن بدلاً من ذلك ، اوضحت هنا طريقة اخرى للحصول على الـ object اللازم . استدع الـ GetDC كـ device context جديدة ، وقم بتمريرها للـ EnumFonts مع الـ nil (parameter لا يستخدم) ، عنوان الـ callback function الخاصة بنا ، و pointer الى الـ StringList لحمل اسماء الـ font .

مرة اخرى ، ان الـ try-finally block مطلوب لضمان ان الـ device context قد تم تحريرها بشكل سليم اذا ما تم حدوث أى exceptions فى استدعاء الـ EnumFonts أو اثناء تنفيذ الـ callback function . ولكن باستعادة الـ Canvas.Handle ، يمكن تبسيط الـ code الى الآتى . لقد كتبت النسخة الاطول لتوضيح الطريقة السليمة للتعامل مع resources النظام مثل device context handles . اذا لم تكن فى حاجة الى فعل هذا ، استخدم هذه النسخة الاقصر :

```
procedure TMainForm.FormCreate(Sender: TObject);
begin
 EnumFonts(Canvas.Handle, nil, @EnumFontsProc,
     Pointer(ListBox1.Items)); // Pass data to callback fn
end;
```

وللمراجعة ، ان الاختلافات الرئيسية بين تقنيات الـ ١٦ بت والـ ٣٢ بت هى :

* يتم تعريف الـ callback function ذات الـ ١٦ بت مع الكلمة الاساسية export ؛ تستخدم النسخة ذات الـ ٣٢ بت sStdcall بدلاً من export .

ان الـ callback function ذات الـ ١٦ بت لا يمكن تمريرها مباشرة للـ Windows function مثل الـ EnumFonts . يجب أولاً ان تنشئ procedure

instanc باستدعاء الـ MakeProcInstance، وتمرر هذه النتيجة الى الـ callback function . يجب ايضاً ان تكون واثقاء من تحرير الـ procedure instance .

* ان الـ Windows 95, 98, & NT يستطيع ان يستدعى functions التطبيق مباشرة، ولاحاجة لاستدعاء الـ MakeProcInstance. قم بتمرير عنوان الـ function كما تم التوضيح فى تطبيق الـEnumfon .

Tip **فكرة:** مازال بامكانك استدعاء الـ MakeProcInstance والـ FreeProcInstance فى تطبيقات الـ ٣٢ بت؛ ولكن، الـ Windows functions هذه تعود الآن دون تنفيذ أى code .ولتحويل الـ ١٦ بت الى ٣٢ بت بسرعة، قد تستطيع ان تستبدل الـ export بـ sStdcall وتعيد الـ compile. لاحظ اننى قلت "قد" . للحصول على افضل النتائج، قم بتحديث برامجك لتمرير عناوين الـ callback function مباشرة كما هو موضح هنا .

يستخدم الـ Enumfon اوامر conditional compilation directives لل compile المختلف إعتماداً على نسخة Delphi التى تستخدمها . ان النسخ المختلفة لـ Enumfonts procedure تكون محتجزة داخل هذه الاوامر، والتى تعتبر نافعة فى إنشاء مشروعات تقوم بـ compile كل نسخ Delphi:

```
{$IFDEF VER80} // Delphi 1.x
...
{$ELSE}     // Delphi 2.x +
...
{$ENDIF}
```

functions التاريخ والوقت:

ان الـ System unit للـ Object Pascal والتى تستخدمها كل التطبيقات تعرف نوع بيانات الـ TDateTime على انه يعادل Float . وهناك مساعدة الـ procedures والـ functions التى تستخدم الـ TDateTime لأى عملية تاريخ ووقت احتاجها.

ان جانب العدد الصحيح لمتغير الـ TDateTime يساوى عدد الايام التى مرت منذ ٣٠ ديسمبر، ١٨٩٩ (هذا يطابق تنسيق تاريخ الـ OLE Variant-تواريخ Delphi 1.0 المحسوبة من عام ١٠٠٠). يخزن الـ TDateTime الوقت على انه

fractional part للمتغير، واليوم يبدأ من منتصف الليل الذى مضى. على سبيل المثال، ٠,٧٥ تمثل الساعة السادسة مساءاً-ثلاثة ارباع اليوم بعد منتصف الليل.

استخدام الـ Trunc function للـ Object Pascal لإستخراج جانب اليوم من قيمة الـ TDateTime. واذا كان لديك إثنين من متغيرات الـ TDateTime وهما Date1 و Date2، فإن العبارة التالية تعين للعدد الصحيح N عدد الايام بين هذين التاريخين. ولان ما يهم هو القيمة المطلقة للطرح، فلا يهم أى التاريخين تعين اولاً:

```
N := abs(Trunc(Date1) - Trunc(Date2));
```

وهناك function تاريخ ووقت اخـــــــرى وهى الـ IncMonth فى الـ SysUtils unit. وهذه الـ function تقوم بادخال قيمة TDateTime مساوية لتاريخ وهو عدد معين من الشهور بعد أو قبل تاريخ محدد. على سبيل المثال، ان تمرير يوم الـ ٣١ من يوليو الى الـ IncMonth وطلب شهر واحد بعد هذا التاريخ يؤدى الى إدخال ٣٠ اغسطس. يمكنك استخدام هذه الـ function لبرمجة الاعمال الشهرية-طباعة تقرير فى آخر كل شهر، مثلاً.

ان الـ IncMonth function، والـ parameters الخاصة بها يتم تعريفها كما يلى:

```
function IncMonth(const Date: TDateTime;
 NumberOfMonths: Integer): TDateTime;
```

- **Date: TDateTime**؛ قم بتمرير أى تاريخ الى هذا الـ parameter، لا يهم الوقت، فقط التاريخ. (استخدم الـ StrToDate function لتحويل التواريخ فى تنسيق الـ string الى قيم TDateTime).

- **NumberOfMonths**؛ قم بتمرير عدد الشهور التى يمكنك بواسطتها زيادة الـ Date. قم بتمرير قيمة سلبية لإنقاص الـ Date.

ان الـ IncMonth function تقوم بارجاع TDateTime object مساوياً لعدد من الشهور بعد أو قبل تاريخ محدد. على سبيل المثال، قم بإضافة ListBox و Button على form، اضغطوا مرتين Button object لانشاء OnClick. اكمل

هذا الـ procedure باستخدام الـ codeالموجودة فى القائمة (٢١-٦)، والموجودة على القرص المدمج فى دليل Source\Misc فى ملف Date1.pas. قم بتشغيل البرنامج واضغط الزر لملئ الـ ListBox بتواريخ من ٣١/ ١/ ٩٧ الى ٣١/ ١/ ٩٩. عند ما تفحص هذه القائمة، فستجد انها تشمل كل تواريخ نهاية الشهور مثل ٢٨/ ٢/ ٩٧ و ٣٠/ ١١/ ٩٨.

القائمة (٢١-٦):Misc\Date1.pas

```
procedure TForm1.Button1Click(Sender: TObject);
var
 D1, D2: TDateTime; // Date variables
    I: Integer;     // for-loop variable
begin
 D1 := StrToDate('1/31/98');
    for I := -12 to 12 do
    begin
     D2 := IncMonth(D1, I);
        ListBox1.Items.Add(DateToStr(D2));
       end;
```

ملحوظة: ان الـ IncMonth function قد تم برمجتها بشكل صحيح للتعامل مع التواريخ فى الالفية القادمة. على سبيل المثال، إنها تدخل بطريقة صحيحة التاريخ ٢٩/ ٢/ ٠٠ لان عام ٢٠٠٠ هو سنة عقدية نادرة، واستخدام الـ IncMonth و functions التاريخ والوقت الأخرى لـ Delphi يمكن ان يساعدك على تجنب الـ "millenium bugs" أو اخطاء الألفية فى تطبيقاتك.

معلومات نوع وقت التشغيل:

يستطيع الـ compiler ان يحدد من الـ source code للبرنامج انواع الـ objects التى يقوم بإنشاءها ولكن بالنسبة للـ objects التى يتم انشاؤها فى وقت التشغيل، قد لا يكون لـ source code متاحة، ويجب ان تحتوى الـ objects على run-time type information (RTTI) أو معلومة نوع وقت التشغيل التى تصف ماهية هذه الـ objects. وهذا موضوع شائك فى البرمجة بالـ

object-oriented. وتوجـد حـاليـاً مـقـاييس قليلة عن افـضل format يمكن استـخـدامـه لتـعـريف objects ووقت التـشـغيل . ان الـ COM objects أى الـ Common Object Model تقـتـرب من هذه المقـاييس ؛ ولكن ليـست كل الـobjects تتطابق مع الـ COM .

وتوجد تعريفات الـ RTTI فى ملف الـ Typinfo.pas الموجودة فى دليل الـ \Source\Vcl\...\ الخـاص بـ Delphi . ولقـد قـامت الـ Inprise Corporation بتحذير المبرمجين من ان تعريفات الـ RTTI سوف تتغير فى إصدارات Delphi المستقبلية . وللتوصل الى تحديث بطريقة اسهل ، احرص على ان تعزل استخدامك عن هذه المعلومة .

على القرص المدمج: وكمثال على كيفية استخدام الـ Typinfo unit ، فإن القرص المدمج يحتوى على Inidata.pas ، تم كتابته بواسطة المراجع الفنى لهذا الكتاب ، وهو Danny Thorpe . وتستخدم الـ Danny unit الـ RTTI object لكتابة قيم خاصية object فى ملفات البدء (ini.) لـ Windows . وباستخدام هذه الـ unit ، يمكنك كتابة كل ما يخص الـ dialog من check box ، نافذة تحرير ، radio button ، controls اخرى للف unit . ، وتقرأ هذه القيم مرة اخرى بسهولة . وها هى بعض الملاحظات من Danny والتى تصف هذه الـ unit :

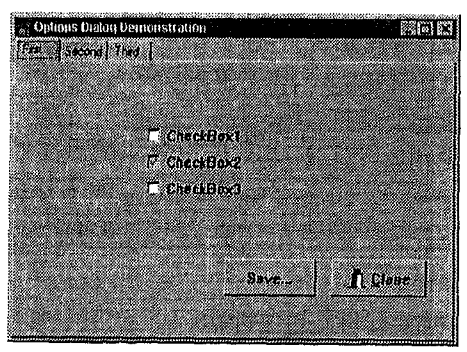
" ان الـRTTI يمكن المطورين من كتابة code تعمل بـ objects باسلوب عام لتشكيل الصفات التى يملكها الـ object بدقة فى وقت التشغيل . ان الـ Compiled code يتم الاشـارة إليـه على انهـا الـ early bound لان كل قـرارات الوصـول والـ link تتم فى وقت الـ compile . ويمكنك الـ RTTI من إنشـاء code تعـتـبـر late bound ، وهذا يعنى ان الوصول والـ link يتم تحديدها فى وقت التشغيل " .

" ان الـ IniData unit يمكن أن تكتب بيانات مستخدم متضمنة لأى form أو component . ان هدف الـ unit هو تحميل وتخزين بيانات المستخدم فقط التى يحتويها الـ component . ولقد تم بناء الـ unit على بعض الاقتراضات التى ليس لها علاقة بالـRTTI-مثل ان الـ component الذى له خاصية واحدة هامة تحتوى على بيانات المستخدم ، وان هذه الخاصية بسيطة (ليست قائمة أو class ، مثلاً) . اذا اردت تخزين كل شئ فى form ، فقط اكتبها فى (stream) " .

On the CD-ROM

على القرص المدمج: لتوضيح كيفية استخدام الـ Inidata.pas، قم بتحميل مشروع الـ Options فى Delphi والذى يوجد على القرص المدمج فى دليل Source\Options. قم بتشغيل البرنامج، واختر عبارته على صفحات الـ TabbedNotebook الثلاث للـ dialog يوضح شكل (٢١-٤) عرض البرنامج. اضغط زر الـ Save لحفظ اختيارات الـ Test.ini فى دليل الـ C:\Windows. إنه البرنامج وأعد تشغيله لتحميل الخيارات من على القرص. ها هو ملف Test.ini قمت بإنشاءه:

```
[TMainForm]
CheckBox1=cbUnchecked
CheckBox2=cbGrayed
CheckBox3=cbChecked
RadioButton1=False
RadioButton2=False
RadioButton3=True
Edit1=Delphi 4 Bible
Edit2=IDG Books Worldwide, Inc.
```



شكل (٢١-٤): يوضح تطبيق الـ Options كيفية استخدام الـ Inidata unit لقراءة وكتابة خصائص الـ object فى ملفات ini.

On the CD-ROM

على القرص المدمج: ان رأس قطاع ملف الـini. تساوى اسم الـ form class وكل عنصر يعتبر component object ذات اسم، وله قيمة تساوى خاصية component واحدة. (يمكنك حفظ خاصية object واحد). ان ملف الـ Main.pas الخاص بالمشروع فى القائمة (٢١-٧)

يوضح كيفية استخدام الـ Inidata unit . يوجد هذا الملف على القرص المدمج فى دليل Source\Options .

القائمة (٢١-٧) : Options\Main.pas. هذا البرنامج يوضح كيفية استخدام الـ Inidata unit، والتى يتم تخزينها فى دليل الـ Source\Initdata

```
unit Main;

interface

uses
 Windows, SysUtils, Classes, Graphics, Controls,
    Forms, Dialogs, Inidata, StdCtrls, Buttons, TabNotBk,
    IniFiles, ComCtrls;

type
 TMainForm = class(TForm)
     TabbedNotebook1: TTabbedNotebook;
        CheckBox1: TCheckBox;
        CheckBox2: TCheckBox;
        CheckBox3: TCheckBox;
        RadioButton1: TRadioButton;
        RadioButton2: TRadioButton;
        RadioButton3: TRadioButton;
        Edit1: TEdit;
        Edit2: TEdit;
        BitBtn1: TBitBtn;
        BitBtn2: TBitBtn;
        procedure FormCreate(Sender: TObject);
        procedure BitBtn1Click(Sender: TObject);
       private
     procedure LoadOptions;
        procedure SaveOptions;
       public
     { Public declarations }
       end;

var
 MainForm: TMainForm;
```

```
implementation

{$R *.DFM}

procedure TMainForm.LoadOptions;
var
 IniFile: TIniFile;
begin
   IniFile := TIniFile.Create('test.ini');
    try
  LoadDataFromINI(MainForm, IniFile);
        finally
     IniFile.Free;
        end;
end;

procedure TMainForm.SaveOptions;
var
 IniFile: TIniFile;
begin
 IniFile := TIniFile.Create('test.ini');
    try
  SaveDataToINI(MainForm, IniFile);
        finally
  IniFile.Free;
        end;
end;

procedure TMainForm.FormCreate(Sender: TObject);
begin
 RegisterINIDataProp('TCheckBox', 'State');
    RegisterINIDataProp('TRadioButton', 'Checked');
    RegisterINIDataProp('TEdit', 'Text');
    LoadOptions;
end;
```

```
procedure TMainForm.BitBtn1Click(Sender: TObject);
begin
 SaveOptions;
end;

end.
```

ان OnCreate للبرنامج يسجل الـ component classes وخاصية واحدة للقراءة والكتابة فى ملف الـ ini. ، على سبيل المثال ، لتسجيل TRadioButton، ينفذ الـFormCreate هذه العبارة :

```
RegisterINIDataProp('TRadioButton', 'Checked');
```

ولان الـ TCheckBox components يمكن ان يكون لهـا ثلاث حـالات ، يجب عليك ان تسجل خاصية الـ states ، وليس الـ Checked . يمكنك ايضاً تسجيل الـEdit باستخدام هذه العبارات :

```
RegisterINIDataProp('TRadioButton', 'Checked');
RegisterINIDataProp('TEdit', 'Text');
```

إختبر الـ LoadOptions والـ SaveOptions procedures كأمثلة على كيفية قراءة وكتابة خصائص الـ object فى ملف الـ ini. . تقوم الـ LoadOptions بإنشاء TIniFile object, ، وتستدعى الـ LoadDataFromINI فى الـ Inidata unit لقراءة الخصائص المحفوظة فى objects تابعة للـ MainForm's . وبطريقة مشابهة تقوم الـ SaveOptions باستدعاء الـ SaveDataToINI لإنشاء أو تحديث ملف الـ ini. .

:Online Help

ان إنشاء ملف online help الخاصة بالـ Windows يبدك هو بالنسبة للتوثيق لـ assembly language بالنسبة للبرمجة ان الخطوات صعبه ، والعملية تمنع الوقوع فى الخطأ ، والنتائج تجعلك تحن للايام التى لم يكن المستخدم يتوقع ان تكون البرمجيات على هذه الدرجة من التعاون .



قبل البدء فى شرح online help ا فإننى ادعوك للبحث عن مولد ملف الـ help يمكن ان ينشئ ملفات الـ help . الخاصة بالـ Windows بطريقة تلقائية ولا استطيع ان ارشح برنامجاً بعينه لأننى لم استخدمها كثيراً. ولكن، هذه البرامج توفر نظام ادخال لإدخال strings البحث، وقيم الفهارس، والكلمات الاساسية، والملاحظات، والعناوين، والنص.

Note **ملحوظة:** ان برمجيات اليوم تضم ملفات html. لتوفير online فى form لنمط الـ World Wide Web page. فى الحقيقة، ان واجهة تطبيق الـ Windows نفسها من المتوقع ان تنجذب ناحية ما يشبة واجهة تطبيق مستخدم الـ Web بغض النظر عن حدوث هذا، فإن صعوبات توفير online لا تقل.

يأتى Delphi بنسخة من الـ Help Workshop, الخاصة بالـ Microsoft's، والتى يمكنك استخدامها لإنشاء ملفات online الخاصة ببرنامجك (فى الـ hlp. format). يوجد البرنامج فى دليل الـ Help\Tools الخاص به فى ملف الـ Hcw.hlp. تصفح هذا الملف لتبدأ فى استخدام الـ Help Workshop. .

أى كانت الطريقة التى تنوى استخدامها، فبعد إنشاء ملف الـ hlp.، قم بتعيين اسم الملف الخاص به لخاصية الـ HelpFile الخاصة بالـ Application object. يجب ان تفعل هذا فى وقت التشغيل (فى الـ form OnCreate مثلاً). إدخل عبارة مثل هذه:

```
Application.HelpFile := 'HelpMe.Hlp';
```

لتقديم مساعدة معينة لـ objects فى form، قم بتعيين قيم help index. وهذه القيم اعداد صحيحة. على سبيل المثال، باستخدام الـ Object Inspector، قم بتعيين ١٠٠٣ لخاصية الـ HelpContext للـ Button يمكنك تعيين قيم help context فى وقت التشغيل، ولكن عادة ما تقوم بهذا عندما تصمم الـ form وعندما تضيف الـ components الى النافذة.

Tip **فكرة:** وعدم الانتظار طويلاً لتعيين قيم help context هو الافضل غالباً. افعل هذا وانت تنشئ forms التطبيق، وحافظ على اثر قيم الـ

context التى تستخدمها . قد تكون القيم سالبة . قد تريد اتباع method نظامى لترتيب الـ indexes . على سبيل المثال ، قم بتعيين مدى محددة مثل ١٠٠٠ . . . . . ١٩٩٩ لـ forms محددة .

ان قيم Help context لا يمكن بسهولة تحديثها ديناميكياً ، ويجب ان تحتفظ بحساب القيم التى تعينها . ومولدى الـ help يقومون بصورة تلقائية بهذا ولكن عندما تقوم بإنشاء online يدوياً ، فإنك تحتاج (spreadsheet) أو قاعدة بيانات لتسجيل الموضوعات ، context-sensitive strings ، قيم context index ، والكلمات الاساسية .

لفتح ملف الـ help لموضوع معين ، استدع الـ HelpContext method للـ Application object . على سبيل المثال ، استخدم عبارة مثل هذه :

```
Application.HelpContext(Button1.HelpContext);
```

أو ، لكى تنتقل الى موضوع indexed بواسطة context-sensitive string ، استدع الـ HelpJump كمايلى :

```
Application.HelpJump('Subject');
```

يقوم الـ HelpJump بارجاع True أو False ، اعتماداً على ما اذا كان الـ Application له ملف help معين . يمكنك عرض رسالة خطأ اذا ادخل الـ HelpJump القيمة بـ False :

```
if not Application.HelpJump('Subject') then
 ShowMessage('Unable to open help file');
```

يجب ان يكون لتطبيقك قيمة HelpContext واحدة على الاقل مساوية الـ Contents page index الخاصة بملف الـ help . اذا لم تعين قيمة غير صفرية لخاصية الـ HelpContext للـ form الرئيسية على الأقل ، لا يستطيع المستخدم ضغط F1 لإظهار الـ context-sensitive online .



ملحوظة: لان الـ packages تقدم امكانية ان تكون الـ dialogs وعناصر واجهة تطبيق المستخدم الأخرى المدرجة فى تطبيق ليس لها وصولاً لنظام الـ help الخاص بالتطبيق ، فقد اضافت الـ Inprise Corporation

خاصية الـ HelpFile للـ TForm فى Delphi 3. عندما تكون نافذة الـ form نشطه ويطلب المستخدم الـ help بضغط F1، تلبى الـ VCL طلب الـ help باستخدام خاصية الـ HelpFile للـ form اذا كانت معينة، والا تستخدم الـ CVL خاصية الـ TApplication.HelpFile لهذا التطبيق.

Dynamic Link Libraries :

ان الـ Dynamic Link Libraries، أو الـ DLL، تعتبر نوعاً من subroutine package التى توجد فى ملف code منفصل، وتستطيع التطبيقات التشارك فيها. ان اغلب مكتبات الـ Windows تم توزيعها مع ملفات الـ DLL، ومكتبات الـ control المخصص يتم تقديمها عادة كـ DLL، ويمكنك تطوير انواع اخرى من المكتبات ذات الـ functions التى يمكن التشارك فيها فيما بين التطبيقات المتعددة.

باستخدام Delphi، يمكنك كتابة واستخدام الـ DLLs الخاصة بك، ويمكنك استدعاء functions فى الـ DLLs المطورة، فى نظم اخرى. يمكنك ان تضع component objects على forms وتضع النافذة فى DLL. والتطبيقات الاخرى، حتى تلك التى لم يتم تطويرها مع Delphi، يمكن ان تستخدم الـ DLL- انها package كامل فى مكتبة ملف code قابلة للتشارك.

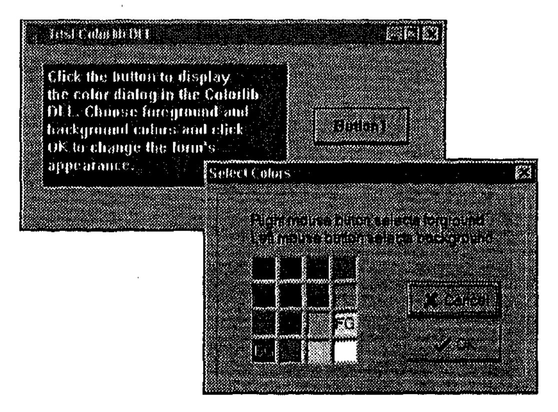
إنشاء الـ DLL:

كمشروع Delphi، تعتبر الـ DLL مكتبة لها procedures و functions يمكن ان تستدعيها التطبيقات الاخرى. فإن كل الاشياء الاخرى داخل الـ DLL مؤمنة وكأنها فى سيارة مصفحة. ولذلك، فإن احد المهام الرئيسية فى إنشاء DLL جديدة هو تعريف كيفية الوصول الى محتوياتها.

ان واحداً من الاسئلة المثارة هو عما اذا كانت الـ DLL تستطيع إنشاء form و component objects اخرى. والإجابة هى، نعم قطعاً ! فى الواقع، ان هذا هو احد الاسباب الرئيسية فى كتابة Delphi DLL. يمكنك ان تضع form و أى Delphi components فى DLL. ان أى برنامج تم كتابته فى أى نظام تطوير Windows آخر يمكن ان يقوم بتحميل الـ DLL لتعرض الـ form. وهذه الطريقة

رائعــة فى دمج امكانات Delphi مع النظم الأخــرى مـثل الـ C ، والـ ++C ، والـ Visual Basic . (ولكن ، الـ ActiveX توفر طريقة اخرى) .

على القرص المدمج: عندما إنشاء DLL جديدة ، اختبر اولاً الـ form والـ components كتطبيق Delphi عام . قم بإنشاء وكتابة واختبار هذا المشروع كـمـا تفعل مع أى مشروع آخر . وكـمـثـال ، فـإن تطبـيق الـ ColorDll على القــرص المدمج فى دليل الـ Source\Colordll ينشئ واجــهــة وخلفية dialog اللون الموضحة فى شكل (٢١-٥) . يوجد dialog باكـملة على DLL منفـصلة تستطيع التطبيقات الاخرى استـخـدامهـا . يوضح الـ objects الـ Delphi DLL والتى يمكن ان تكـون لـهـــا objects form وأى component اخرى والتى يمكنك استخدامها فى التطبيقات instances .



شكل (٢١-٥): يعرض الـ Colorlib DLL displays اختيار اللون هذا، باستخدام الـ ColorGrid object.

على القرص المدمج: كل الملفات التالية موجودة على القرص المدمج فى دليل الـ Source\Colordll . انسخ الدليل على مـحــرك القــرص الصلب الخــاص بك ، ثم افــتح مـشـروع الـ Colorlib فى Delphi . اضغط Ctrl+F9 للـ compile DLL . وهذا البرنامج غير قابل للتنفيذ ، ومحاولة تشغيله بضغط F9 تؤدى الى رسالة خطأ .

لكتـابة البـرنامج ، قـمت اولاً بتطوير dialog اللون على انه تطبيق Delphi عام . وعندما انتهيت ، قمت .

بتعديل ملفات الـ source code كمايلى يمكنك استخدام هذه الخطوات لتحويل أى مشروع Delphi الى DLL :

١- اجعل اسم الـ form unit بـ ClrForm والمشروع بـ ColorLib ، احفظ المشروع كما تفعل فى المعتاد .

٢- قم بتعريف procedure أو function فى قطاع واجهة التطبيق لـ form unit- وليس فى الـ form class . يقوم مستخدموا الـ DLL باستدعاء هذا (subroutine) للتمكن من الوصول الى المكتبة . يمكنك تعريف أى عدد من نظم الوصول كما تحتاجه الـ DLL الخاصة بك . على سبيل المثال ، ان الـ Clrform.pas تعرف الـ function FBGetColors كمايلى . وبما ان لغات اخرى يمكن ان تستدعى الـ function ، قم بتعريف انواع بيانات الـ Windows فقط (الـ WordBool مثلاً ، بدلاً من الـ Pascal's Boolean) . إتبع التعريف التالى بالكلمة الاساسية export ، التى تخبر الـ compiler ان يجعل هذه لـ function متاحة لمستخدمى الـ DLL :

```
function FBGetColors(var FColor, BColor: TColor):
      WordBool; export;
```

٣- قم بتنفيذ procedure أو function الوصول . توضح القائمة (٢١-٨) النتيجة النهائية لـ dialog اختيار اللون . سوف اشرح هذا الجزء من البرنامج بعد هذه الخطوات .

٤- بعد ان تنتهى من كتابة الـ form unit ، قم بتحويل مشروع التطبيق الى DLL . لتفعل هذا ، اختر امر الـ View|Project Source لفتح ملف مشروع dpr. انتقل الى هذه الصفحة فى محور نص الـunit ، وقم بالتغييرات التالية .

٥- قم بتغيير الـ program الى library .

٦- احذف Forms من امر الـ uses . (تحتاج المشروعات الى هذه الـ unit لاستدعاء الـ Application methods. والـ DLLs ليست تطبيقات ، وملفات مشروعاتها لا تحتاج الى Forms unit.

٧- فيـمـا بين امـر الـ Resource {$R} والسطر الاخيـر من امـر الـ uses للمشروع، اضف كلمة exports، متبوعة باسم procedure أو function وعلى سبيل المثال FBGetColors unit. يمكنك إدخال واحدة أو اكثر من اسماء النظم الفرعية هنا.

٨- احذف كل العبارات فيما بين الـ begin والـ end، تاركاً code block خالياً. توضح القائمة (٢١-٩) النتيجة النهائية لمشروع الـ ColorLib.

٩- اضغط Ctrl+F9 لـ compile وإنشاء ملف الـ dll code. . لا يمكنك تشغيل الـ DLL، لذا اذا ضغطت F9، فإنك تتلقى رسالة خطأ. ولكن هذا لا يمنع compile الـ DLL ولا يسبب ضرراً.

القائمة (٢١-٨): Colordll\Clrform.pas.

```
unit Clrform;

interface

uses
 Windows, SysUtils, Classes, Graphics, Controls,
    Forms, Dialogs, StdCtrls, Buttons, ColorGrd, ExtCtrls;

type
 TColorForm = class(TForm)
     ColorGrid: TColorGrid;
        Label1: TLabel;
        Label2: TLabel;
        CancelBitBtn: TBitBtn;
        OkBitBtn: TBitBtn;
        Bevel1: TBevel;
        procedure CancelBitBtnClick(Sender: TObject);
        procedure OkBitBtnClick(Sender: TObject);
       private
    { Private declarations }
       public
  { Public declarations }
       end;
```



```
var
 ColorForm: TColorForm;

function FBGetColors(var FColor, BColor: TColor):
WordBool;
 export;

implementation

{$R *.DFM}

{ Exit dialog via Cancel button }
procedure TColorForm.CancelBitBtnClick(Sender:
TObject);
begin
 ModalResult := mrCancel;
end;

{ Exit dialog via Ok button }
procedure TColorForm.OkBitBtnClick(Sender: TObject);
begin
 ModalResult := mrOk;
end;

{ Get foreground and background colors }
function FBGetColors(var FColor, BColor: TColor):
WordBool;
begin
 Result := False;
    ColorForm := TColorForm.Create(Application);
    try
     if ColorForm.ShowModal = mrOk then
        with ColorForm do
        begin
         FColor := ColorGrid.ForegroundColor;
            BColor := ColorGrid.BackgroundColor;
            Result := True;
           end;
 finally
  ColorForm.Free;
```



```
    end;
  end;

  end;
```

القائمة (٢١-٩): Colordll\Colorlib.dpr. لتحويل مشروع تطبيق الى DLL، قم بتعديل ملف الـ dpr الخاص به كما هو موضح هنا

```
library Colorlib;

{ Important note about DLL memory management:
ShareMem must be
 the first unit in your library's USES clause AND your
   project's (select View-Project Source) USES clause if
   your
   DLL exports any procedures or functions that pass
   strings as
   parameters or function results. This applies to all strings
   passed to and from your DLL-even those that are nested
   in
   records and classes. ShareMem is the interface unit to
   the
   BORLNDMM.DLL shared memory manager, which
   must be deployed
   along with your DLL. To avoid using
   BORLNDMM.DLL, pass string
   information using PChar or ShortString parameters. }

uses
// ShareMem,  { Not required in this example }
 SysUtils,
   Classes,
   Clrform in 'CLRFORM.PAS' {ColorForm};

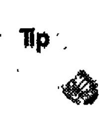
exports
 FBGetColors;

{$R *.RES}
```

```
begin
end.
```

ان النسخ الحديثة من Delphi تقدم مشروع DLL يمكنك استخدامه لإنشاء DLLs جديدة من لاشئ. استخدم ...File|New، اضغط الـ New page tabe، اختر الـ DLL Block. بالرغم من ان اتباع الخطوات الموضحة هنا يتطلب مزيداً من العمل، الا اننى قمت بتطوير الـ DLLs كتطبيقات مستقلة لأغراض اختيارية، ثم حولت الـ code الى الـ DLLs لمشاركتها فيما بين التطبيقات.

فى الـ unit الخاصة بالـ DLL، وهى Clrform.pas، الـ function الـ FBColors تظهر كيفية قيام الـ DLL بإنشاء الـ form object. تقوم الـ function بارجاع الـ WordBool (أو نوع بيانات الـBoolean الخاص بالـ Windows) لتشير اذا ما كان المستخدم قد اغلق الـ dialog بضغط OK (True) أو Cancel (False). أولاً، تبدأ الـ function قيمتها المدخلة بـ False. بعد ذلك، تنشئ ColorForm object باستدعاء الـ Create method للـ TColorForm class's. ولأنك قد حولت التطبيق الى DLL، وهو لم يعد ينشئ form بصورة تلقائية، ويتعين على الـ function ان تفعل هذا تحت تحكم البرنامج.



فكرة: استخدام دائماً الـ try-finally block كما توضح الـ Clrform.pas لتضمن أنه حتى اذا حدث exception، يتم حذف الـ form object.

وداخل الـ try block ، يستدعى البرنامج الـ ShowModal method الخاص بالـ ColorForm's ، والذى يدخل القيمة المعينة للـModalResult بواسطة OnClick للـ OK Button والـ Cancel Button للـ dialog's. اذا رجعت الـ ShowModal بـ mrOk، يعين البرنامج الوان المتغيرة للـ function. هذا يؤدى الى تمرير معلومة الـ dialog خلفاً الى البرنامج وكعلامة على النجاح، يعين البرنامج True كنتيجة لك الـfunction.



تحـذير: اذا قامت الـ DLL الخاصة بك بتمرير أى string parameters أو نتائج للـ function، أو اذا استخدمت arrays



ديناميكية، فيجب عليك استخدام الـ ShareMem unit فى الـ uses directive للتطبيق والـ DLL. ولا يحتاج البرنامج الاختبارى الموضح هنا الى هذه الـunit، ولكننى أضفتها كتعليق لاحدد مكانها. وهذه القاعدة تنطبق حتى على الـ strings المحاطة فى سجل أو class object اذا قمت بتمرير بيانات strings بأى وسيلة من والى الـ DLL، اضف ShareMem لعبارة الـ uses كما هو موضح هنا. تذكر ان تفعل هذا فى كلاً من الـ DLL والتطبيق الذى يستخدمها.

يقــوم الـ finally block باســتــدعــاء الـ Free method الخــاص بالـ ColorForm's لتدمير الـ form class وكل الـ components. هذا يعيد ذاكرة النظام الى حالتها فى بداية الـfunction. يجب ان تكون بالغ الحرص فى التعامل مع الـ DLL عند إنشاء وتحرير الـ objects بطريقة سليمة.

ولضمان السلامة، يجب ان يتم السيطرة على الأخطاء فى الـ DLL من خلال try-except block. هذا يمنع هروب أى exceptions من نطاق الـ DLL، وهو امر خطير لان معالجة الـ exceptions أمر غير قياسى خارج حدود التطبيق. فلا احد يستطيع ان يتوقع ما قد يحدث للـ exception إذا خرج من نطاق الـ DLL وراح يتجول فى الـ Windows. وإحدى الطرق لمعالجة الاخطاء الخطيرة هو ان يستدعى برنامج معالجة الـ exceptions الخاص بالـ DLL لتستدعى function مقدمة من البــرنامج وتســتــخــدم الـ DLL. والـcallback function هذه يمكن ان تحــدث exception التطبيق. وهذا الحل يحافظ على الـ exception فى مداها-وتبقى الـ DLL exceptions فى الـ DLL؛ وتبقى exception التطبيق فى التطبيق.

استخدام الـ DLL:

ان لديك الآن Colorlib.dll فى دليل. بعــد ذلك، يمكنك إنشــاء تطبــيق يستدعى function الخاصة بالـ DLL لعرض dialog اختيار لون. ولمجرد الهو، يقوم التطبيق بتلوين واجهة الـ Memo الخاص به باستخدام قيم اللون المدخلة.



على القرص المدمج: يمكنك إنشاء التطبيق بنفسك أو تحميل مشروع الـ Testdll من نسخة لدليل الـColordll على القرص المدمج. توضح القــائمــة (٢١-١٠) الـ source code للـTestdll.pas، والتى توضح كيف يقوم تطبيق Delphi بتحميل واستخدام الـ DLL.

القائمة (٢١-١٠): Colordll\Testdll.pas. هذا البرنامج يوضح كيفية تحميل واستخدام الـ Colorlib DLL من الفصل السابق

```
unit Main;

interface

uses
// ShareMem, { Not required by this example }
 Windows, SysUtils, Classes, Graphics, Controls,
    Forms, Dialogs, StdCtrls;

type
 TMainForm = class(TForm)
     Button1: TButton;
        Memo1: TMemo;
        procedure Button1Click(Sender: TObject);
       private
  { Private declarations }
       public
  { Public declarations }
       end;

var
 MainForm: TMainForm;

implementation

{$R *.DFM}
{ Declare external function in the Colorlib.Dll }
function FBGetColors(var FColor, BColor: TColor):
WordBool;
far;
 external 'Colorlib';
```



```
{ Activate color dialog in the Colorlib.Dll }
procedure TMainForm.Button1Click(Sender: TObject);
var
 FColor, BColor: TColor;
begin
 FColor := Font.Color;  { Form's text color }
    BColor := Color;     { Window background color }
    if FBGetColors(FColor, BColor) then { Call DLL ! }
    begin
     Memo1.Color := BColor;    { Assign window color }
        Memo1.Font.Color := FColor; { Assign text color }
       end;
end;

end.
```

فى التطبيق المضيف (التطبيق الذى يستخدم الـ DLL)، قم بتعريف نفس الـ function أو الـ procedure الذى تعرفه الـ DLL unit. ولكن، كما يوضح الـ Testdll.pas، استخدم form مختلفة بعض الشئ. قم بتعريف function. من الناحية الفنية، لا يجب عليك ان تفعل هذا اذا كنت واثقاً ان الـ code سيتم عمل compile لها باستخدام نسخة Delphi ذات ٣٢ بت، ولكننى ضمنت الكلمة الاساسية far على أية حال. كذلك بتعريف الـ function بـ external، يتبعها أسم الـ DLL string (ناقص امتداد الـ dll.). هذا يخبر الـ compiler ان البرنامج سوف يقوم بإدخال الـfunction.

والخطوة الاخيرة هى اسهل الخطى. استدع الـ access function كما يوضح OnClick للزر الخاص بالبرنامج. تفتح الـ DLL نافذة الـ dialog form، وتتحكم تماماً فى كل الـ component objects. اذا رجعت الـfunction بـ True، يستخدم البرنامج قيم اللون المختار والتى تم تمريرها الى الخلف عبر اثنين من الـ TColor parameters المتغيرة.

الـ Message Event Handlers والـ Application:

تقدم الـ TApplication class عدداً من الـ events والتى يمكنك ان تكتب لها event handlers. ولان الـ Application object يكون غير مرئى فى الـ form، والـ TApplication غير متاحة على الـ VCL palette، فيجب عليك استخدام تقنية مختلفة لإنشاء الـ event handlers هذه. إتبع هذه الخطوات :

* بتعريف الـ event handlers فى القطاع الـ public أو الـ private للـ form class (غالباً الـ Private). لا تقم بتعريف الـ event handlers قبل الكلمة الاساسية private حيث يولد الـ Delphi التعريفات والـ component object والـ methods .

* قم بتنفيذ الـ event handlers بكتابة procedure. اجعل اسم الـ procedure مسبوقاً باسم الـ form class. وبوجه عام، استخدم نفس الـ format الذى يستخدمه Delphi عندما ينشئ الـ event handlers للـ objects التى تختارها باستخدام الـ Events page tab لنافذة الـObject Inspector .

* قم بتعيين الـ event handlers لخاصية الـ Application event : OnActivate، OnDeactivate، OnException، OnHelp، OnHint، OnIdle، أو OnMessage .

الـ event handlers للـ TApplication:

اتبع هذه الخطوات كمثال على تعيين للـ Application event handlers. يوضح البرنامج تقنية مفيدة للغاية يمكنك استخدامها لاداء عمليات خلفية دون استخدام الـTimer :

١- قم بإنشاء تطبيق جديد أضف Label و Memo على الـform .

٢- اضف التعريفين التاليين للقطاع الـ private للـ TForm1 class وتعتبر X هى عدد صحيح (والتى تم بدؤها بصفر عند إنشاء الـform ويقوم AppOnIdle event handler بزيادته وعرضه فى الـ Caption الخاص بالـ Label. يجب ان يكون للـ event handler الـ parameters و الـform العامة الموضحة هنا، ولكن يمكنك ان تسميه ماتشاء-ابحث فى events معينة عن الـformats المطلوبة لـ :handler procedures الخاصة بهم :

```
dX: Integer;
procedure AppOnIdle(Sender: TObject; var Done:
Boolean);
```

٣- قم بتنفيذ الـ AppOnIdle procedure بإضافة السطور التالية فى قطاع الـ implementation للـunit . لاحظ ان اسم الـ form class يســبق اسم الـ procedure وتفصلها نقطة . يقوم الـ event handler بزيادة متـغير الـX الخاص وعرضه بتحويل الـ X الى string وتعيين النتيجة للـ Caption الخاص بالـLabel :

```
procedure TForm1.AppOnIdle(Sender: TObject; var Done:
Boolean);
begin
 Inc(X);
   Label1.Caption := IntToStr(X);
end;
```

٤- لربط الـ event handler مع الـ OnCreate event الخــاص بالـ TApplication ، قم بتـعـيين اسـم الـ procedure الخـاص بالـ event handler الى event . يمكنك ان تفعل هذا فى أى مكان فى البرنامج ، ولكن غالباً ما تقوم به فى الـ form OnCreate . اختـر الـ form واضـغط مـرتين قـيـمـة الـ OnCreate eventالخاص بها لإنشاء هذا الـ event handler ، وإدخل عبارة التعيين التالية :

```
Application.OnIdle := AppOnIdle;
```

٥- قم بتشغيل البرنامج ، والذى يقوم بتحديث الـ Label باستمرار ، حتى إثناء ما تقوم بادخال نص فى نافذة الـ Memo .

Note

ملحوظة: بالرغم من اننى استخدمت Label لهذا العرض ، الا إن تغيير الـ Label caption يجعل نافذة الـ Label غير صـالحـة ، مما يؤدى الى رسـالة wm__Paint تنهى حـالة الـ idle مـبكراً . وهناك انشطة اخـرى قليلة الحدوث وتشـمل استـدعـاء MessageBeep، كتابة مخرجـات للملف ، أو ارسال رسالة الى تطبيق آخر .

يمكنك ان توقع برسالة Windows بانشاء event handler للـ Application.OnMessage . هذا يساعدك على تحسين استجابات التطبيق على الرسائل المختلفة، أو تقديم خدمات لرسائل غير معروفة. قم بتعريف OnMessage event handler كمايلي :

```
procedure AppOnMessage(var Msg: TMsg; var Handled:
Boolean);
```

- **Msg:** هو سجل بيانات من نوع الـ TMsg يحتوى على handle لنافذة، وقيمة الرسالة، وparameters، ووقت ارسال الرسالة، واحداثيات الـ cursor . وهذه القيم تعتمد على الرسالة المحددة التى تم تلقيها.

- **Handled:** اذا كان OnMessage الخاص بك يتعامل مع رسالة، حدد هذا الـ parameter بـ True . اذا لم تكن تتعامل مع رسالة، أو اذا اردت ان تستعيد اعماله الافتراضية، حدد الـ Handle بـ False.



تحذير: تمر عشرات الآلاف من الرسائل فى كل ثانية عبر الـ TApplicationOnMessage. كن شديد الحذر فيما تفعله فى الـ event message handler.

Subclassing لرسائل الـ Windows:

هناك طريقة اخرى للتعامل مع رسائل الـ Windows وهى بكتابة method indexed لثابت رسالة معينة. على سبيل المثال، يمكنك استخدام هذه التقنية للتقسيم الفرعى لاستجابة عنصر القائمة على اختبار أمر. تستطيع form البرنامج ان تنتظر رسالة الأمر، تؤدى عمليات إضافية عندما يأتى الأمر على الخط، وتمرر الرسالة لأداء نشاطها المعتاد.

جرب هذه الخطوات للتقسيم الفرعى لأمر الـ File لقائمة Exit ، وعرض رسالة لها ازرار Yes و No يمكن للمستخدمين ضغطها لإنهاء البرنامج اختيار الأمر :

١- ابدأ تطبيقاً جديداً. أضف الـ MainMenu . اضغط الـ object مرتين، واستخدم الـ Menu Designer الخاص بـ Delphi's لإنشاء قائمة File ذات أمر Exit .

٢- اختـر أمر الـ File|Exit في الـ form وأضف Close في الـ Command event handler.

٣- قم بتشغيل البرنامج، واختر الـ File|Exit لأختبار ان هذا الامر يغلق نافذة البرنامج.

٤- إنك تقـوم الآن بالتـقـسيم الفـرعى للـ event handler للأمـر الـ Exit بالامـسـاك برسـالة الـ wm__Command Windows الخـاصـة به، والتى تم اصدارها عندما اخترت الأمر. إدخل تعريف الـ message event handler التالى فى القطاع الـ public أو الـ protected للـ TForm1 class. ان البـرنامج له نفس اسم الرسالة، ناقص الـ underscore فى الثابت من الاسم. يتلقى الـ procedure متغير TWMCommand الذى يحتوى على معلومات حول هذا النوع من رسالة الـ Windows:

```
procedure WMCommand(var Message: TWMCommand);
 message wm_Command;
```

٥- قم بتنفيذ الـ WMCommand كمايلى. إدخل الـ procedure فى قطاع الـ implementation للـ unit:

```
procedure TForm1.WMCommand(var Message: TWMCommand);
begin
 if Message.ItemId = Exit1.Command then
   begin
  if MessageDlg('Quit program?', mtConfirmation,
      [mbYes, mbNo], 0) = mrNo then
       begin
      Message.Result := 1;  { Message was handled }
        Exit;      { Exit without calling inherited }
       end;
     end;
// inherited WMCommand(Message); { Old style }
 inherited;     { Call inherited method }
   Message.Result := 1;  { Indicate message was handled }
end;
```

٦- قم بتشغيل البرنامج، واختر الـ File|Exit . اذا أجبت بـ Yes ينتهى البرنامج، اذا اجبت بـ No، يستمر البرنامج. وهذا النوع من الاستفسار يمكن ان يقدم بطرق أخرى، ولكن اهمية التقنية الموضحة هنا هى ان الـ form class تعترض طريق رسالة الـ Windows لتقدم برمجة اضافية، أو لتحل محل اعمال الرسالة .

ان الـ event handler لرسالة الـ WMCommand يفحص حقل الـ ItemId فى سجل الـ Message. وهذا الحقل يساوى قيمة الأمر، والتى يتم تعيينها بصورة تلقائية لخصائص الـ TMenuItem.Command عندما تصمم Menu. استخدم خاصية الـ object لتجد هذه القيمة كما هو موضح هنا للـ tExit1.Command .

لعمل رسالة معترضة، استدع الـ inherited كما تم التوضيح هنا فى القسم السابق لـ Delphi، كان من الضرورى ان تستدعى procedure موروثاً بالاسم. وهذا لم يعد مطلوباً-استخدم فقط الكلمة الاساسية inherited كما فى المثال. ان استدعاء الـ inherited إختيارى، ولكن ان لم تفعل هذا، يمكنك ان تمحو مستوى كامل من العمليات لرسالة معينة. اذا اردت ان تستبدل تماماً استجابة الـ code لرسالة معينة، إذن لا يجب عليك استدعاء الـ inherited. ولكن عادة يجب ان تفعل هذا:

Note

ملحوظة: ان الـ event handler للـ Windows مثل الـ WMCommand يتم بتعريفهما الآن بشكل خاص الى TForm-class مثلاً. والـ classes المشتقه لا تقوم بإستدعاء أو الإشارة الى التعريفات الخاصة للـ class السالفة. على المستوى الفنى، هذا هو السبب فى انك يجب ان تستخدم الـ inherited بلا argument كما كان مطلوباً فى الماضى.

حدد حقل الـResult للـ Message parameter's بصفر لتشير الى ان الرسالة لم يتم التعامل معها. حدد الـResult بواحد اذا تعاملت مع الرسالة. فى هذه الحالة، لا تهم قيمة الـResult لان الـprocedure الموروث يحددها غالباً .

يمكنك استخدام التقنية الموضحة هنا للسيطرة على أى الـ event handler فى class . انظر التعريف الموجود فى ملف واجهة تطبيق الـ unit (Forms.pas،

```
            ShowMessage('Nonclient left button click'); //
            Obsolete!
             wm_NCRButtonDblClk:
           ShowMessage('Nonclient right button click');
            end;
      inherited WndProc(Message);
  end;

  end.
```

قم بتنفيذoverridden window procedure باختبار الـ Msg فى متغير الـMessage. ان الـ Msg يساوى ثابت رسالة الـ Windows.

وغالباً، سوف تستخدم عبارة case، كما فى القائمة لتقارن الـ Msg مع واحدة أو أكثر من رسائل الـ Windows-فى هذه الحالة، رسالتى الضغط المزدوج فى الـ Client-area. تذكر ان واحدة فقط منهما هى التى تعمل-ان الـ wm__NCLButtonDblClk مهجورة الآن، على الأقل لاغراض الـ subclassing للـ event. استدع الـ WndProc لعمل رسائل أخرى، أو يمكنك ان تحدد الـ Message.Result بقيمة غير صفرية وتستدعى الـ Exit لتشير الى انك قد تعاملت مع الرسالة. ولكن يجب ان تمرر اغلب الرسائل على الـ method الموروث للعمل الأفتراضى.

الـFile Streams:

يشمل Delphi نظام object-oriented file-streaming والذى يمكنك استخدامه لقراءة وكتابة البيانات، أو الـ components، وobjects اخرى فى ملفات القرص. توضح القائمة (٢١-١٢) كيفية استخدام الـ TStream والـclasses TFileStream لكتابة Button object على ملف قرص، ثم قراءته بعد ذلك. يوضح التطبيق ايضاً كيفية إنشاء component object تحت التحكم التام للبرنامج. قم بتشغيل البرنامج ثم اضغط Create لإنشاء زر وعرضه على الـ form اضغط Write لتسيير الزر الى ملف القرص المسمى Test.stm (يمكنك ان تجعل اسم الملف كما تشاء، ولا يجب ان يكون امتداد اسم الملف stm.) ان ضغط الـ Write يدمر الـ button object ويمحو من النافذة. اضغط Read لقراءة الـ object مرة اخرى من الـ file stream .

ان الـ Main unit تحدد الـ TButton object ، B، بالسير من والى ملف قرص. انظر هذا التعريف فى قطاع الـ unit implementation. للتحديد على الشاشة ما اذا كان الزر آتياً من برنامج أو من stream، يعرض اثنان من BOnClick1 و BOnClick2 رسائل مختلفة باستخدام الـ ShowMessage procedure. يقوم الـ Procedure EnableButtons بتشغيل وابطال ازرار الـ Create والـ Write و الـ Read لاقتراح نظام صحيح من العمليات. وهذا على سبيل المثال يمنع إنشاء أو قراءة أكثر من Button object واحد فى كل مرة.

القائمة (٢١-١٢): Compstrm\Main.pas

```
unit Main;

interface

uses
 SysUtils, WinTypes, WinProcs, Messages, Classes,
Graphics, Controls,
 Forms, Dialogs, Buttons, StdCtrls;

type
 TMainForm = class(TForm)
    CreateButton: TButton;
      WriteButton: TButton;
      ReadButton: TButton;
      CloseBitBtn: TBitBtn;
      procedure FormCreate(Sender: TObject);
      procedure CreateButtonClick(Sender: TObject);
      procedure WriteButtonClick(Sender: TObject);
      procedure ReadButtonClick(Sender: TObject);
      procedure FormDestroy(Sender: TObject);
     private
    { Private declarations }
      procedure EnableButtons;
      procedure BOnClick1(Sender: TObject);
```

```
      procedure BOnClick2(Sender: TObject);
       public
     { Public declarations }
       end;

var
 MainForm: TMainForm;

implementation

{$R *.DFM}

const
 fileName = 'Test.stm'; { The stream's filename }

var
 B: TButton; { Object to be streamed to and from disk }

{ OnClick event handler before writing Button to stream }
procedure TMainForm.BOnClick1(Sender: TObject);
begin
   ShowMessage('Thanks. I needed that!');
end;

{ OnClick event handler after reading Button from stream }
procedure TMainForm.BOnClick2(Sender: TObject);
begin
 ShowMessage('Hello. It''s nice to be back!');
end;

{ Enable and disable buttons depending on whether the
Button
object and Test.Strm file exist }
procedure TMainForm.EnableButtons;
```

```
begin
 CreateButton.Enabled := False; { Disable all buttons }
    WriteButton.Enabled := False;
    ReadButton.Enabled := False;
    if B nil then
  WriteButton.Enabled := True  { Write if B exists }
       else begin
     CreateButton.Enabled := True; { Create if B doesn't
       exist}
        if FileExists(fileName) then
         ReadButton.Enabled := True { Allow read if file
          exists }
           end;
end;

{ Initialize other buttons to control clicking order }
procedure TMainForm.FormCreate(Sender: TObject);
begin
 EnableButtons;
end;

{ Create Button object entirely under program control }
procedure TMainForm.CreateButtonClick(Sender:
TObject);
begin
 B := TButton.Create(Self);  { Create the Button object }
    B.Parent := Self;       { Assign parent object }
    B.Left := 240;        { Assign left position }
    B.Top := 72;         { Assign top position }
    B.Caption := 'Click me!';  { Assign label }
    B.OnClick := BOnClick1;  { Assign first event handler
   }
    EnableButtons;        { Update operation order }
end;
```

```
{ Write Button object to file stream, and then destroy it }
procedure TMainForm.WriteButtonClick(Sender: TObject);
   var
 Stream: TStream;
begin
{ Create a new disk file stream }
 Stream := TFileStream.Create(fileName, fmCreate);
    try
     Stream.WriteComponent(B);  { Write Button object }
        B.Free;           { Destroy Button object }
        B := nil; { Prevent form's OnDestroy from Freeing
      again }
        EnableButtons;        { Update operation order }
       finally
  Stream.Free; { Destroy the file stream object }
       end;
end;

{ Read Button object from stream }
procedure TMainForm.ReadButtonClick(Sender: TObject);
var
 Stream: TStream;
begin
{ Create stream object in read-only mode }
 Stream := TFileStream.Create(fileName, fmOpenRead);
    try
     B := TButton(Stream.ReadComponent(nil)); {Read &
      Create B}
        B.Parent := Self;     { Always assign parent! }
        B.OnClick := BOnClick2;  { Assign second event
      handler }
        EnableButtons;      { Update operation order }
       finally
```



```
        Stream.Free; { Destroy the file stream object }
         end;
    end;

    { Destroy button if user closes app before completing test }
    procedure TMainForm.FormDestroy(Sender: TObject);
    begin
     if B nil then B.Free;
    end;

    { Register the TButton class with Delphi stream system }
    begin
     end.
```

ان CreateButtonClick ينشئ button object جديد باستدعاء الـ Create constructor لـ TButton class's وتحدد ناتج الـ object Reference لـ B . لكى تجعل الـ object مرئياً، فيجب ان يكون له Parent، وهو فى هذه الحالة الـform الرئيسية . ان اغلب الـobjects تحتاج الى قيم Left و Top لتحديد موضعها- يمكنك ايضاً تحديد الـ Width والـHeight للـ object، ولكن هذه الخصائص والخصائص الاخرى لها قيم افتراضية . وحتى يفعل الزر شيئاً عندما تضغطه ، يعين الـ BOnClick1 procedure للـ Button OnClick .

ان الـWriteButtonClick يوضح كيفية إنشاء file stream جديد . افعل هذا باستدعاء الـ TFileStream Create constructor مع arguments، اسم ملف وثابت نمط مثل fmCreate، واحفظ الـ object Reference الناتج فى متغير TStream أو TFileStream .

جدول (٢١-١): يعرض الثوابت التى يمكن تمريرها إلى TFileStream

Constant	الأثر
O fmOpenRead	يفتح ملف للقراءة
fmOpenWrite	يفتح ملف للكتابة
fmOpenReadWrite	يفتح ملف للقراءة والكتابة

fmShareExclusive	يفتح ملف للاستخدام المختصر للتطبيق
fmShareDenyWrite	يمنع تطبيقات اخرى من الكتابة على الملف
fmShareDenyRead	يمنع تطبيقات اخرى من القراءة من الملف
fmShareDenyNone	يسمح لتطبيقات اخرى من القراءةوالكتابة على الملف
fmCreate	ينشىء ملف جديد ويفتحه للكتابة

استخدم تعبيرات الـ OK المنطقية لدمج ثوابت نمط الملف. على سبيل المثال، يفتح التعبير التالى ملفاً للقراءة والكتابة، ولكن يمنع التطبيقات الأخرى من الكتابة على الملف مادام مفتوحاً:

```
fmOpenReadWrite or fmOpenDenyWrite
```

استدع الـ WriteComponent method الخاص بالـ TStream ولكتابة object على الـ stream. يمكنك كتابة أى عدد تحتاجه من الـ components، ويمكن ان يكون من نفس النوع أو انواع مختلفة. بعد كتابة الـ Button، يقوم التطبيق بهدمه ويحدد B، بـ nil. اذا اغلقت التطبيق، فإن هذا يمنع الـ form OnDestroy من تحرير نفس الـ object مرة أخرى، مما قد يسبب General Protection Fault (GPF).

ان الـ WriteButtonClick يستخدم الـ try-finally block لضمان ان الـ Stream قد تم تحريره حتى اذا حدث exception. استخدم دائماً الـ try-finally بهذه الطريقة فى الـ procedures والـ functions الخاصة بك التى تنشىء الـ TFileStream object.

ان الـ ReadButtonClick Procedure يقرأ الـ streamed object من على القرص. ومرة اخرى، تستدعى الخطوة الأولى الـ Create constructor الخاص بالـ TFileStream ولكن هذه المره باستخدام نمط الـ fmOpenRead. وهذه العبارة تقرأ الـ object على الـ file stream وتعينه الى B:

```
B := TButton(Stream.ReadComponent(nil));
```

إستدعى الـ TStream ReadComponent method لقراءةcomponent أو file stream . قم بتـمـرير مـرجع الى الـ ReadComponent component لقراءة object فى object موجودة بالفعل، أو تمريره كما فى هذا المثال لكى تجعل الـ ReadComponent ينشئ object جديد تماماً ويرجع الـ ReadComponent نوع الـ TComponent، لذا يجب عليك استخدام type-cast لتعيين النتيجة الى نوع معين من مرجع الـ component مثل B .

بعد قراءة object من على الـ stream، تأكد من ان تجعل له Parent- فهذه المعلومـة ليست مـحفـوظة فى الـ stream لانه form مـختلفة أو نفس الـ form فى موضع مختلف فى الذاكرة قد تملك الـ object . يجب عليك ايضاً ان تعيد تعيين أى event handlers وكـقـاعـدة عـامـة، أى مـعلومـة عن عنوان مـعين (event handlers، مـشيـر لـ object اخرى، وخاصيـة الـ Parent) قد لا تكون على الـ stream لان هذه المعلومة تكون مختلفة بعد قراءة الـ objects .

قبل ان يستطيع التطبيق قـراءة وكتـابة الـ object فى file stream ، يجب عليـه لن يسـجل objects classes مع أسلوب الـfile streams الخـاص بـ Delphi . يمكنك ان تفعل هذا فى الـ form OnCreate، ولكن لضمـان ان العمل قد تم باسرع ما يمكن، اضف الكلمة الاساسية initialization قبل عبارة الـ end فى الـ unit . وفيمـا بينهمـا، أضف استدعاءات للـ RegisterClass لكل class تكون هذه الـ module Register classes object أو stream objects .

تطبيقات الـ Internet:

ان طبـعـة الـ Client/Server من Delphi تأتى بمجـمـوعـة كـاملة من الـ components لتطبيق تطبـيـقـات الـ Internet والـ Intranet . (ان الإنتـرنت هى شـبكة تشـبـه الـ Internet فى الـ protocols وتنظيـمـهـا، ولكنهـا مـقـصـورة على المجـمـوعـة المحلية من الحـاسـبـات-مـثـلاً، كل النظم فى شـركـة مـا) . وهذه الـ components تعـتـبـر نافـعـة، ليس فـقط للوصـول الى مـواقع الـ Internet والـ Intranet، ولكن ايضاً لتطوير برمجيات قاعدة البيانات (Remote-access) . على سبيل المثال، Inprise Corporation uses تستخدم Delphi components فى الاخبار عن الأخطاء فى النسخ والذى اصبح متاحاً لمختبرى البرمجيات .

الـInternet components:

يوفر Delphi مجموعة من الـ components لبرمجة الـInternet والـintranet . نسخة الـClient/Server تقدم components إضافية كما هو موضح في شكل (٢١-٦) . ولان قليل من قراء هذا الكتاب سوف يحصلون على نسخة الـ Client/Server الباهظة الثمن ، فإنني اقدم هنا الـ components المقدمة مع الـDelphi Professional فقط .

شكل (٢١-٦): الـ Internet components لطبعة الـ Client/Serve من Delphi

ملحوظة: ان Internet component Inprise Corporation قد أجرى لها بعض التغييرات منذ الاصدار الأول . ان الـ components FTP، NNTP وcomponents الاخرى المؤسسة على الـ ActiveX- قد تم استبدالها في Delphi 4 بالـ VCL components الاصلية المسارية لها والتي تم إنشاؤها بواسطة NetMasters . وهذا التغيير لا يجب ان يؤثر على اسماء الـ classes المستخدمة في النسخ الأولى ، ولكن؛ ولكن نسخة قبل الاصدار الخاصة بي من Delphi لم تكن تشمل كل الـ classes ، بالرغم من ان الـ Inprise Corporation اكدوا لي ان الاصدار الاخير من الـ Internet components classes سيكون مطابقاً للنسخ الاولى .

ان الـ Internet components الخاصة بـ Delphi ليس لها صور بصرية وقت التشغيل ، وهي موضحة كايقونات في نافذة الـ form وتعتبر كل الـ ActiveX controls محاطة باغلفة Delphi component . وبحسب ترتيب ظهورها على الشاشة ، فإن الـ Delphi Internet ActiveX omponents هي :

- **FTP (Internet File Transfer Protocol):** ينقل الملفات والبيانات بين الحاسوب المحلي والبعيد .
- **HTML (Hypertext Markup Language):** يقدم مستعرض وثيقة HTML . ويمكن ايضاً ان يستخدم للـ parsing ، وليس لعرض ، أوامر وثيقة الـ HTML .

- **HTTP (Hypertext Transport Protocol)**: يسترجع وثائق الـ HTTP دون تصفح أو إمكانات عمل الصورة.
- **NNTP (Network News Transfer Protocol)**: يقدم لمجموعات الاخبار قراءة وارسال الاخبار.
- **POP (Post Office Protocol)**: يحصل على البريد من الـ UNIX أو انواع اخرى من الـ servers التى تؤيد الـ POP3.
- **SMTP (Simple Mail Transfer Protocol)**: يتوصل الى الـ SMTP mail servers ويقدم ارسال البريد.
- **TCP (Transmission Control Protocol)**: لتطبيقات الـ Client/Servr التى تحتاج خدمات شبكة الـ TCP.
- **UDP (User Datagram Protocol)**: ينفذ الـ WinSock لتطبيقات الـ Client/Servr . يمكن ايضاً ان يرسل ويستقبل بيانات UDP.

تطبيقات الـ Internet الخاصة بـ Delphi:

يقدم Delphi العديد من التطبيقات التى توضح كيفية استخدام الـ Internet components . وكمقدمة سريعة من برمجة الـ Internet ، جرب متصفح الـ Web الموجود فى دليل الـ Demos\Internet\Html لـ Delphi's.

يوضح هذا البرنامج كيفية إنشاء متصفح Web كامل باستخدام HTML component. واحد فقط . قم بتحميل ملف مشروع الـ Webbrows.dpr على هذا الدليل فى Delphi's ، اضغط F9 للـ compile والتشغيل يوضح شكل (٢١-٧) نافذة البرنامج بعد اتصالها لموقع الـ Web الخاص بالـ Borland's.

لطلب وثيقة HTML على الـ Internet ، قم بإضافة HTML component على form ، وقم ببرمجة الـ event handler لاستدعاء الـ RequestDoc method للـ component كمايلى :

```
HTML1.RequestDoc(URLs.text);
```



شكل (٢١-٧): يوضح الـ Web الخاص بـ Delphi's برمجة الـ Internet مع الـ components

يفـتـرض أن الـ URLs هي ComboBox component أو component آخر له خاصية نص محددة بالـ URL الذى تريده . يمكنك ايضاً تمرير string حرفى للـ RequestDoc:

```
HTML1.RequestDoc('www.inprise.com');
```

يعـتـبـر الـ HTML component object مـقيـداً فى الحـصـول على نص المصـدر لوثيـقـة الـ HTML . لكى تفـعل هذا، اتصل بالموقع باسـتـدعـاء الـ RequestDoc، ثم قم بالوصول الى خاصية الـ SourceText لوثيقة الـ HTML . وهذ هى قائمة string تحتوى على نص من الوثيقة .

ان متصفح الـ Web الخاصة بـ Delphi يستخدم خاصيـة الـ SourceText لعرض مصـدر الـ HTML فى نافذة، مضيفاً النص الى Memo يستخـدم هذه العبارات :

```
Memo1.Lines.Clear;
Memo1.Lines.Add(HTML1.SourceText);
```

وهناك اثنين من الـ HTML events تعتبر مفيدة فى استكشاف بداية وتوقف استرجاع الوثيقة . اختر HTML object واضغط باب الـ Events على الـObject Inspector . ثم اضغط مرتين الـ OnBeginRetrieval والـ OnEndRetrieval لإنشاء events handlers لكل منهما وهما :

- **OnBeginRetrieval**: يتم استدعاؤه عند بداية تحميل الوثيقة .
- **OnEndRetrieval**: يتم استدعاؤه عند إنهاء تحميل الوثيقة .

ويستخدم متصفح الـ Web هذين الـ events لتشغيل وابطال زر الـ Cancel للبرنامج . (من الواضح ان عملية الـ Cancel غير لازمة بعد ان يتم تحميل الوثيقة كاملة) . يمكنك ان تفكر فى استخدامات اخرى مفيدة لهذين الـ events- عرض رسالة على الشاشة تخبر المستخدمين ان الوثيقة يتم تحميلها ، مثلاً .

وبالطبع يمكنك استخدام العديد من الـInternet components ، العديد من الـ events والخصائص الاخرى لتطوير تطبيقات الـ Internet .

مزيداً من الافكار:

إننى اجد ان الافكار التالية ، والتى تنساب فى ملحوظات مع بعض ملفات مجموعة الـ DOS ، نافعة للغاية اثناء استخدام وتعلم Delphi . وقد تجدها كذلك انت ايضاً .

إنقاص حجم ملف الـ code:

فيما يلى إثنين من التقنيات الهامة التى تستطيع ان تقلل حجم ملف الـ code بدرجة كبيرة :

* إغلق معلومة ازالة الاخطاء ، والتى يتم كتابتها على ملف الـ exe . اثناء debugging sessions . أختر ...Project|Options واضغط الـ Linker page tab . اغلق الـ TD32 debug info check box . هذا يمنع الـ compiler من نقل معلومات ازالة الخطأ الى ملف الـ code .

* استخدم packages وقت التشغيل . يقوم Delphi بإنشاء ملف exe . كامل يحتوى على code تطبيقك زائد كل الـ components code على الـ VCL .

هذا يجعل ملف الـ code ملائماً للتوزيع لأنه يجب عليك تقديم ملف واحد فقط للمستخدم النهائى. ولكن النسخ المتعددة من ملف code التطبيق تحتوى على نسخ لا داعى لهـا من الـ VCL. قم باستئصال هذه النسخ الزائدة باختيار ProjectǀOptions...، اضغط الـ Packages page tab، وقم بفتح الـ check box الذى يحمل بطاقة Build مع packages وقت التشغيل. وهذا الاختيار البسيط يمكن ان يقلل حجم الـ code الخاص بك بـ ٥٣٢ك أو أكثر. ولكن، لان الـpackages يجب ان تكون متاحة لتشغيل البرنامج، فأى محفوظات لمساحة القرص يتم تحقيقها فقط اذا كان لديك تطبيقات Delphi متعددة على نفس النظام.

لكلا الـ methods، اعد بناء تطبيقك لإنقاص ملف exe. التام. من المهم ان تعيد البناء بحيث يتم إعادة compile كل الـ methods لا تضغط فقط F9 للـcompile والتشغيل للتطبيق كما تفعل عادة. لكى تنشئ ملف exe. التام، اختر ProjectǀBuild.

ان packages وقت التشغيل ايضاً تقلل استخدام الذاكرة. لان الـ packages ليس أكثر من مجرد compiled DLL بشكل خاص، فإن الـ code تكون مشتركة بين كل تطبيقات Delphi فى حالة تشغيل.

Note

ملحوظة: عند استخدام الـpackages، التى يجب ان توفرها للمستخدمين النهائيين-على سبيل المثال، ملف VCL40.bpl فهذا الملف وملفات الـ bpl. الاخرى المطلوبة يجب ان يتم نسخها على دليل الـ C:\Windows\System الخاص بالمستخدم النهائى.

: Focus changes

فى بعض نسخ Delphi، لا نوصى بأداء أى اعمال تؤدى الى focus change فى الـ event handler والذى يكون هو نفسه جزء فى الـ focus change. على سبيل المثال، ان محاولة عرض الـ message dialog فى الـ Edit OnExit، يمكن ان يجعل البرنامج يفقد القدرة على التحكم فى controls اخرى.

وهذه المشكلة لم تعد موجودة فى Delphi 3والنسخة التالية. لتعرف اذا كان نظامك يعرض هذا السلوك، ضع بعض الـ Edit objects على form واستدع الـ

OnExit ShowMessage لـ object واحـد. قم بتشـغيـل البرنـامج وانتـقل من controls الى controls بعـد ان تغلق الـ message dialog، قـد يكون لديك مـشكلة فى نقل الـ forms الى controls اخـرى- باعطاء الـ dialog يعطى الـ control التالى الـ focus، وهو ما تفعله الـ OnExit عندما تعرضـه. على سبيل المثال، ان ضـغط الـ Tab بعـد اغـلاق الـ dialog لم يحـرك الـ forms الى نافـذة الـ Edit التالية كما هو متوقع. وهذه المشكلة ليس خطيرة.

Multiple program instances :

كـمـا تعـرف، يمكنك تشـغـيل instances مـتـعـددة من لبـعض برامج الـWindows، ولكن بعضـها يقيدك بان تقوم بتشغيلها مرة واحدة فقط. ويشترط الـ Microsoft ان الـ instances الجديدة لتطبيق ما يتم إنشائها عندما يقوم المستخدم بتشغيلها لثانى مرة أو أكثر. ولكن هذا الاشتراط يمكن ان يؤدى الى مشاكل كبيرة. حـاول تشـغـيل اثنين من الـ Microsoft Word instances -عندمـا فـعـلت هذا، تلقيت رسالة الخطأ ان وثيقة الـNormal.dot يتم استـخدامه بواسطة تطبيق آخر وغالباً، هذه هى الحالة الاصلية للـ Web.

وبدلاً من ذلك، يجب ان تقرر هل من الملائم لتطبيقك ان يتم تشغيله على انه حـالة فـردية، أو على انه حـالات مـتعـددة عندمـا يعيـد المستخدمون بدء تشـغيل البرنامج. والحيلة فى عـمل هذا ليس بسيطة ولكن ممكنة بواسطة تعـديل الـ code source لمشروع Delphi. اتبع هذه الخطوات لتجرب التقنية :

١- ابدأ تطبيـقـاً جـديداً تريد ان تحفظ ملفـاته على القرص بحيث تستطيع تشغيل ملف الـ exe. من الـ Windows Explorer. (لا يمكنك تشغيل program instances من داخل Delphi). قم بإنشاء دليل للمشروع. احفظ ملفات الـ unit والمشروع باستخدام اسماء الملفات الافتراضية لها.

٢- اختـر View|Project Source لفـتح مشـروع البرنامج كمـا ترى، ان المشروع بالفعل يعتبر برنامج الـ Pascal الرئيسى للبرنامج.

٣- أضـف Dialogs و Windows units لإمـر الـ uses للملف المشـروع. لقد فعلت هذا تحت الـ Forms unit مباشرة، ولكن الموضع لا يهم غالباً.

٤- قم بتعديل عبارات المشروع بين الـ begin والـ end باستخدام القائمة (٢١-١٣) كمرشد لك ان الـ code المذكورة تفترض ان الـ form الرئيسية للتطبيق تحمل اسم TForm1 . اذا غيرت اسم الـ form ، يجب ان تقوم بتحديث كل المراجع المشيرة إليها .

٥- احفظ المشروع لتنشئ الدليل الخاص به ، واضغط Ctrl+F9 للـ compile . اغلق كل الملفات . انتقل الى الـ Windows Explorer ، وقم بتشغيل ملف الـ Project1.exe . دون اغلاق النافذة ، قم بتشغيل البرنامج مرة اخرى - فى هذه المرة سوف ترى رسالة ان البرنامج فى حالة تشغيل الآن . اغلق dialog الرسالة للعودة بصورة تلقائية الى حالة البرنامج الاولى .

Note

ملحوظة: وجدت انه يجب ان اغلق ملفات التطبيق قبل ان اتمكن من تشغيل اثنين من حالات البرنامج . وقد يكون هذا نتيجة لاستدعاء الـ FindWindow عندما كانت ملفات مصدر التطبيق مفتوحة فى Delphi لان class النافذة للـ form الموجودة فى Delphi تحمل اسم TForm1 .

القائمة (٢١-١٣): استخدم هذه القائمة كمرشد لك لمنع مشروع من التشغيل عدة مرات

```
program Project1;

uses
 Forms,
    Dialogs,
    Windows,
    Unit1 in 'Unit1.pas' {Form1};

{$R *.RES}

begin
 HPrevInst := FindWindow('TForm1', nil);
    if HPrevInst = 0 then
    begin
     Application.Initialize;
```

```
          Application.CreateForm(TForm1, Form1);
            Application.Run;
           end else
        begin
      ShowMessage('Application is already running');
            Windows.SetFocus(HPrevInst);
            Windows.SetForegroundWindow(HPrevInst);
           end;
    end.
```

فى الماضـــــى ، كـــــان الـ Windows يحـــــــدد الـ HPrevInst للـ program instance handle الموجـودة بالفعل . لم يعـد هـذا مـوجـوداً ، ولكن ، يمكننا ان نجعل الخطوة مزدوجة باستدعاء الـ FindWindow كما تم التوضيح . ان الـ parameter الاول الذى يتم تمريره الى الـ FindWindow هو اسم class والـ parameter التالى ، والذى يستخدم هنا ، يمثل Caption النافذة . هذا يفترض ان اسم الـ class يعرف التطبيق مطلقاً وان الـ form object لم يتم إنشاؤه بعد بواسطة التطبيق .

ويعدل المشروع المعدل الـ HPrevInst ، والتى اذا كانت مساوية لصفر تشير الى ان هذه هى المرة الاولى التى يتم فيها تنفيذ البرنامج . ان تشغيل التطبيق يبحث عن النافـذة ، ويحـدد الـ HPrevInst بـ instance handle التطبـيق . اذا كـان الـ handle غـير صـفـرى ، فـإن عـبـارات الـ else الخاصـة بالمشـروع تعـرض رسـالة وتستدعى الـ Windows API SetFocus function لإعطاء الحالة الاولى input focus . ان عـــرض نافـــذة الحـــالة يتطلـب ايضـــاً اســـتـــدعـــاء الـ SetForegroundWindow كما هو موضح .

لاحظ ان الـ SetFocus والـ SetForegroundWindow يتم استدعائهما بالاشارة مباشرة الى الـ Windows unit ، حيث يتم تعريف كلتا الـ functions . وهذا ليس ضـرورياً ، ولكن لان Delphi components أخـرى لهـا SetFocus methods ، فأن تسبق الاستدعاءات بالـ Windows يضمن ان البرنامج يستدعى الـ methods الصـحيح . يمكنك ان تفعل هذا مع أى unit لها methods مزدوجة وأسماء object متداخله .



فكرة: للانتقال بهدوء الى program instance الاولى ، احذف عبارة الـ ShowMessage .

بعض ملفات مجموعة الـ DOS النافعة:

هل مازالت تكتب اوامر على الـ DOS prompt كما افعل انا عادة ؟ إننى اكتب دائماً ملفات مجموعة batch files لأداء عمليات متنوعة فى نافذة DOS . وفيما يلى بعد الملفات التى وجدتها نافعة للغاية وبخاصة مع Delphi .

على القرص المدمج: توضح القائمة (٢١-١٤) ملف الـ Clean.bat من على القرص المدمج فى دليل الـ Source . فيما عدا الملفات الهامة فى مشروع Delphi . تعتبر القائمة (٢١-١٥) جزءاً من ملف الـ Cleanall.bat الذى يمكنك تشغيله لحذف الملفات التى لاداعى لها بعد ان تقوم بـ compile البرامج المثالية لهذا الكتاب . لاستخدام هذه البرامج ، انسخها على الدليل الذى يحتوى على الادلة الفرعية لمشروعك . قم بتشغيل الـ Cleanall لمسح كل الادلة . أو ، قم بتشغيل الـ Clean لمسح ملفات فردية مثل هذا .

القائمة (٢١-١٤): Clean.bat

```
@echo off
rem
rem CLEAN.BAT
rem Delete files created by Delphi
rem Called by CLEANALL.BAT
rem Input %1==subdirectory where clean.bat is stored
rem
rem ex. call clean Polyflow
rem
if "%1"=="" goto ERROR
if exist %1\NUL goto CONTINUE
goto ERROR
:CONTINUE
cd %1
echo Cleaning %1
```

```
rem
rem Add new files to delete here:
rem
if exist *.bak del *.bak
if exist *.~* del *.~*
if exist *.dsk del *.dsk
if exist *.dsm del *.dsm
REM if exist *.exe del *.exe
rem
rem Go back to directory that has clean.bat
rem
:REPEAT
cd ..
if exist clean.bat goto END
goto REPEAT
:ERROR
echo.
echo Error cleaning: %1
echo No such directory
pause
:END
```

القائمة (٢١-١٥)؛Clean.bat (القائمة جزئية-الباقى على القرص)

```
@echo off
rem
rem cleanall.bat-call clean.bat for all source directories
rem
echo working...
if exist *.bak del *.bak
call clean aboutex
call clean addpage
call clean barchart
call clean bitview
call clean bitview2
```

```
...
echo done
:END
```

لضـمان السـلامـة ، لقـد اشـرت بتـعلق الى التـعليـمـات الموجـودة فى الـ Clean.bat والتى تمسح ملفات الـ exe. اذا قمت بتشغيل هذا السطر ، كن شديد الحذر من ان تمسح كل التطبيقات الموجودة على محرك قرصك الصلب. لا يجب ان تقوم بتثبيت الـ Clean أو الـ Cleanall فى دليل الـ Path العمومى. كذلك قد تريد ان تحتفظ بنوع أو آخر من الملفات. انظر الباب الاول ، فصل "امتدادات اسم الملف" لتعرف وصف الملفات التى يقوم Delphi بإنشاءها.

الـ parameters الافتراضية:

يضيف Delphi 4 افتراضية الى الـ parameters Object Pascal. وهذه الميزة ، والمقتبسة من الـ C++ ، توفر قيم افتراضية للـ function والـ procedure parameter على سبيل المثال ، افترض ان لديك procedure له ثلاثة parameters معرفة كمايلى :

```
procedure CallMe(A: Integer; B: Double; C: String);
```

لاستدعاء هذا الـprocedure فى عبارة program ، يجب ان توفر قيم argument ، الـparameters الثلاثة :

```
CallMe(10, 3.14159. 'Message from a flounder');
```

ولكن ، افترض ان بعض الاستدعاءات للـ procedures لم تكن تحتاج الى توفير قيم للـ parameter ، فبدون الـ parameters الإفتراضية ، يجب عليك كتابة code مثل هذه :

```
CallMe(10, 2.2, '');
CallMe(20, 3.3, '');
CallMe(30, 4.4, '');
```

فى كل حالة ، إنك تحتاج أن تقدم null string للـ CallMe ، بالرغم من أن قيم الـ parameter الذى يتم استخدامه. إن قيم الـ parameter الإفتراضية تساعد على تبسيط هذا النوع من الـ code. قم بتعريف parameter إفتراضى كما يلى :

```
procedure CallMe(A: Integer; B: Double; C: String = '');
```

إن القيمة الإفتراضية المعنية للـ parameter C تخبر الـ compiler أن يستخدم هذه القيمة اذا لم يتم توفير قيمة واضحة. مع هذا التعريف فى موضعه، يمكن أن يتم الآن كتابة الاستدعاءات الثلاثة السابقة للـ procedure كما يلى:

```
CallMe(10, 2.2);
CallMe(20, 3.3);
CallMe(30, 4.4);
```

إن القيم الإفتراضية يمكن تعيينها لأى parameter، ولكن يجب أن تأتى فى آخر التعريف. لا يمكنك مثلاً استخدام تعريف مثل هذا:

```
procedure X(A: Integer = 0; B: Double); // ???
```

هذا لا يؤدى الى الـ compile لأن قيم الـ parameter الإفتراضية يجب أن تأتى فى النهاية دائماً. فى هذه الحالة، يمكنك أن تعكس ترتيب الـ parameters أو توفر قيمة إفتراضية للـ B أيضاً:

```
procedure X(A: Integer = 0; B: Double = 3.14159):
```

إن الـ parameters الإفتراضية تبسط بعض أنواع العبارات، وتساعد على ضمان أن يكون لكل الـ parameters قيم معروفة. والعيب الوحيد من استخدام قيم الـ parameters هى الطريقة التى تشوش بها على الـ arguments الفعلية التى يتم تمريرها الى functions و procedures. على سبيل المثال، إن الاستدعاءات الثلاثة السابقة للـ CallMe لا تعطى اشارة عن أن null string argument قد تم تمريرها كـ argument ثالثة سرية.

:Method overloading

يضيف Delphi 4 ميزة أخرى معروفة ومستعارة من الـ -C++ وهى الـ method overloading أو التحميل الزائد للـ method. هذا يسمح لإثنين أو أكثر من الـ procedures أو الـ functions أن يكون لهم اسماءاً متطابقة، ولكن يعملوا كنظم فرعية مختلفة والقاعدة البسيطة هى: أن الـ overloaded method يجب أن

يختلف فى نوع parameter واحد على الأقل . ومادمت تلتزم بهذه القاعدة ، يمكنك أن تزيد تحميل نفس الـ method كلما دعت الحاجة .

إن زيادة تحميل الـ method يساعدك على إنشاء code مفهومة تتوافق مع الاهداف المختصة بالـ object ، مثلاً ، تطوير مجموعة من الـ procedures كلها تحمل اسم ShowItem والتى تعرض نوع من الشكل البيانى . يمكنك عندئذ كتابة عبارة مثل :

```
ShowItem(X);
```

حيث أن الـ X يمكن أن يكون لها شكل معروف لأنواع عديدة مختلفة . وهذا يعتبر أسهل من كتابة procedures متعددة تحمل اسماء ShowIntegerItem ، ShowGraphicsItem ، ShowImageItem ، الى آخره .

مثال على التحميل الزائد للـ Method:

وكمثال آخر أكثر اكتمالاً ، افترض أنك تريد تطوير procedure لبدء arrays ديناميكية من أنواع مختلفة . أولاً ، إنك تعرف أنواع الـ array الديناميكية كما يلى :

```
type
 TIntArray = array of Integer;
    TFloatArray = array of Double;
    TStringArray = array of String;
```

بعد ذلك ، قم بتعريف متغيرات للـ arrays الثلاثة :

```
var
 IntArray1: TIntArray;
    FloatArray1: TFloatArray;
    StringArray1: TStringArray;
```

وجزئيتى الـ code هذه توضحان الاستخدام العملى لتعريف أنواع الـ array . إننا نحتاج الى إنشاء arrays فعلية للأنواع الثلاثة ، ونحتاج الى تعريف procedure parameters لنفس هذه الأنواع . ويتم تعريف الـ parameters الثلاثة هكذا (داخل قطاع الـ class public مثلاً) :

```
public
procedure InitIntArray(A: TIntArray; V: Integer);
procedure InitFloatArray(A: TFloatArray; V: Double);
procedure InitStringArray(A: TStringArray; V: String);
```

إن كل procedure يتم تنفيذه لتعيين قيمة V لكل موضع array ، وبهذا يحدد كل عناصر الـ array بقيمة معروفة . يمكن تنفيذ الـ procedure الأول كما يلى :

```
var
 I: Integer;
begin
 for I := 0 to High(A) do
     A[I] := V;
end;
```

بعد كتابة هذا الجزء من الـ code ، يمكنك استدعاء procedures إنشاء الـ array مثل هذه :

```
InitIntArray(IntArray1, 0);
InitFloatArray(FloatArray1, 1.0);
InitStringArray(StringArray1, 'Msg:');
```

هذا يحقق الهدف ولكنه فوضوى بعض الشئ مع الـ method overloading ، يمكن تحديد اسم procedure واحد لكل أنواع عند انشائها الـ array . لكى تفعل هذا ، قم بتغيير كل اسماء الـ procedure حتى تصبح واحدة ، وأضف الكلمة الأساسية overload بعد التعريف :

```
procedure InitArray(A: TIntArray; V: Integer); overload;
procedure InitArray(A: TFloatArray; V: Double); overload;
procedure InitArray(A: TStringArray; V: String); overload;
```

ان procedure الثلاثة من الممكن ان يكون لها نفس الاسماء لان تعريفها تختلف فى نوع procedure واحد على الاقل . قم بتنفيذ كل procedure (يجب ان تقوم بتنفيذ الـ procedures الثلاثة منفصلة) كما سبق ، باستخدام نفس

التعـريفـات الموضــحة هنــا . لاستـدعاء الـ procedures ، استـخدم code مثل هذه :

```
InitArray(IntArray1, 0);
InitArray(FloatArray1, 1.0);
InitArray(StringArray1, 'Msg:');
```

لايوجـد سـحر فى هذا الأمـر . فـداخليـاً ، مـازال هنـاك ثلاثة procedures منفصلة . ولكن ، بدلاً من التـمييز بينهـا بالاسم ، مع الـ method overloading ، من الممكـن التــفــرقــة بين الـ procedures بواسطة البـيــانـات التى تم تمريرهـا كـ arguments هذا يوضح الـ code ، ويجـعل من السـهل إضــافــة InitArray procedures جديدة لأنواع بيانات اخرى .

Note

ملحــــوظـة: ان الـ overloaded method يجـب ان يكـون لـه نـوع parameter واحد على الأقل يختلف عن الـ methods الاخرى التى لها نفس الاسـم أو يجـب ان يكـون الـ methods مـــخــتـلفــــة من الـ parameters . ان نوع وعدد الـ parameters هو الذى يهم .

التحميل الزائد للـ method والـ parameters الافتراضية:

يمكنك استخدام قيم parameter افتراضية مع الـ overloaded methods . على سبيل المثال ، يمكن تعيين قيـمة افتراضـية للـ V parameter فى التعـريفـات السابقة باستخدام تعريفات مثل :

```
procedure InitArray(A: TIntArray; V: Integer = 0); overload;
procedure InitArray(A: TFloatArray; V: Double = 1.1);
overload;
procedure InitArray(A: TStringArray; V: String = '');
overload;
```

والآن ، من الممكن بدء الـ arrays للانواع المعرفة بعبارات مثل هذه :

```
InitArray(IntArray1);
InitArray(FloatArray1);
InitArray(StringArray1);
```

ان التحميل الزائد للـ methods، مع قيم الـ parameters الافتراضية، ينتج عادة code مثل هذه، بسيطة وواضحة. ومن الاسهل ان تتذكر اسماء الـ method دون الاضطرار الى مراجعة تعريفها.

على الجانب الآخر، ان دمج الـ Method overloading مع قيم الـ parameter الافتراضية يمكن ان يؤدى الى code محيرة:

```
procedure F(Height: Integer; D: Double = 0.0); overload;
procedure F(Width: Integer); overload; // ???
```

هذا يضع الـ compiler فى حيرة لأنه ليس من الممكن دائماً تحديد أى procedure تنوى العبارة استدعاؤه هذه العبارة، مثلاً:

```
F(10); // ???
```

لا يتم الـ compile. وبسبب قيمة الـ procedure الافتراضية فى التعريف الأول، يمكن ان تستدعى العبارة أى من الـ procedures. للقضاء على المشكلة، يجب عليك الا تعين قيمة procedure افتراضية للـ D، أو يجب ان تعيد تسمية الـ procedure الثانى (بـ F2، مثلاً).

: Method overloading and class constructors

ان الـ Method overloading له قيمة عالية فى إنشاء class constructors. فى النسخ السابقة من الـ Object Pascal، كان يجب تسمية الـ constructors باسماء مختلفة. على سبيل المثال، يمكن لـ class نموذجية فى الـ VCL ان تعرف الـ constructors التاليين:

```
constructor Create(AOwner: TComponent); override;
constructor CreateNew(AOwner: TComponent; Flag:
Integer);
 virtual;
```

مع الـ method overloading، يمكن ان للـ constructor الثانى ان يسمى Create لان له parameter واحد زائد عن الـ constructor الأول. لزياد تحميل كلا الـ constructors، قم بتغيير إسمهما لـ Create وأضف الكلمة الاساسية overload. والتعريفات الجديدة تصبح:

```
constructor Create(AOwner: TComponent); overload;
override;
constructor Create(AOwner: TComponent; Flag: Integer);
 virtual; overload;
```

هذا يؤدى الى تبـسـيط إنشــاء الـ class objects لان الـ constructors الصـحيح يتم الآن بناء على استـخدامه بدلاً من الاسم الظاهرى له. على سبيل المثال. هذه العبارة تنشىء objects من الـ class :

```
Obj1 := TForm.Create(AOwner);
Obj2 := TForm.Create(AOwner, 1);
```

ان العـبـارة الأولى تسـتـدعى الـ constructor الأول، والذى يسـيطر على Create method (overrides) الموروث الخاص به. والعبارة الثانية تستدعى الـ constructor الثانى لأنها تقدم argument عدد صحيح للـ Flag parameter. الخاص بهذا الـ constructor.

: Method overriding versus overloading

ان احدى الصعوبات التى قد تواجهها مع الـ overloaded methods هى عندما يخفى تعريف الـ methods الاصلى والموروث له نفس الاسم. ولنقل هذا بعبارة اخرى، هناك صراع قوى بين الـ overriding على الـ methods موروث والـ overriding للـ methods. ان online help الخـاصـة بـ Delphi تعطى امـثلة مشابهة لمايلى. تقوم code بتعريف classes. ها هى الأولى :

```
FirstClass = class
public
 procedure X(I: Integer); virtual;
end;
```

والـ class الثانى مشتقة من الأولى :

```
SecondClass = class(FirstClass)
public
 procedure X(S: string);
end;
```

وهذه الانواع من التعريفات يتم استخدامها عامة لاستبدال X) procedure في المثال) في class مشتقة ان البرنامج المستبدل في الـ SecondClass يعرف parameter من نوع مختلف عن الـ methods الموروث الخاص به. ولذلك، فإن الاستدعاء الأول فقط من الاستدعائين التاليين للـ X يتم الـ compiles :

```
var
 AnObject: SecondClass;
begin
 AnObject.X('Testing');
    AnObject.X(123); // ???
end;
```

ويفشل الاستدعاء الثانى للـ X في الـ compile لان إعادة التعريف للـmethod الموروث يخفى التعريف الأصلى الذى تعرف على أنه Integer parameter . والـ Method overloading تجعل من الممكن حل هذا النوع من الصراعات. يتم تعريف الـ classes الجديدة كمايلى :

```
FirstClass = class
public
 procedure X(I: Integer); virtual; overload;
end;
SecondClass = class(FirstClass)
public
 procedure X(S: string); overload;
end;
```

ان الاختلاف الوحيد هو استخدام الكلمة الاساسية overload. ولان الـ procedures يعتبران overload فان استدعاءالـ procedure للـ X الآن يتم الـ compile لان الـ compiler يمكن أن يميز بين الـ procedure من خلال انواع الـ parameter المختلفة لهما.

ولان وراثة method وتغيير الـ parameters الخاصة به، وليس اسمه، يؤدى الى إخفاء التعريف الاصلى- في هذا المثال، X في الـ SecondClass يخفى



الـ X فى الـ FirstClass-فإن الـ compiler يصـدر تحـذيراً. لإزالة هذا التحـذير ، استخدم الكلمة الاساسية reintroduce فى التعريف التالى هكذا :

```
SecondClass = class(FirstClass)
public
 procedure X(S: string); reintroduce;
end;
```

هذا يخبر الـ compiler انك تنوى التعتيم على الـ X الاصلية وانك لا تتوقع ان يقوم التعريف الجديد لـ overload الـ method .

: Method overloading and forward declarations

ان كلمـة التـعـريف الاسـاسـيـة forward والخاصـة بالـ Object Pascal يتم استـخدامـها الآن بطريقة مـختلفة قـليلاً بما ان اللغة method overloading. فى الماضى، كـان يمكنك ان تعـريف procedure أو function forward ، مما يشـير الى ان تنفيذها يأتى بعد ذلك فى ملف الـ source code على سبيل المثال،

```
// Declare forward procedure
procedure X(D: Double); forward;
... // Code may call X here
// Implement the procedure
procedure X;
begin
 // Code for X
end;
```

ان تعـريف الـ forward يعطى الـ compiler كل المعلومـات التى يحتـاجـهـا ليتعامل مع العبارات لإستدعاء الـ X. ولكن، اذا كنت تنوى ان overload الـ X بـ procedure ثانى متميز يسمى X ايضاً دون parameters، فإن الـ code السابقة تكون محيرة. ان إضافة الكلمة الاساسية overload للتعريف الاول يجعل الـ compiler ينشئ procedure جديد منفصل يحمل اسم X وبلا parameters ، وهو ليس ما تريد بالطبع. لحل هذا الصراع، يسمح الـ Object Pascal الآن لك

بإعادة تعريف الـ forward procedure مع parameters . على سبيل المثال، لـ overload الـ X مع نسخة بلا parameters، ولتنفيذ نسخة X مع الـ Double parameter، يمكنك كتابة تعريفات مثل هذه :

```
// Declare overloaded forward procedure
procedure X(D: Double); overload; forward;
...
// Implement an overloaded, distinct procedure
procedure X; overload;
begin
end;
// Implement forward declaration
procedure X(D: Double); overload;
begin
end;
```

هذا لن يكون ممكناً اذا لم تكن تنفيذات الـ forward لا يمكن ان يكون لها تعريف للـ parameter . ان الـ X الذى ليس له parameters فى هذا المثال هو procedure منفصل عن الـ X forward procedure الذى يعرف الـ Double .parameter.

arrays ديناميكية:

يضيف Delphi 4 arrays ديناميكية للـ Object Pascal . ان الـ array الديناميكى يعرف نوعه، وليس حجمه . الـ array الفعلى يتم إنشاؤه فى وقت التشغيل بعبارة برنامج . إنشئ array ديناميكياً مع الـ static array العام للـPascal . على سبيل المثال، ان تعريف المتغير هذا :

```
var
 IntArray : array[1 .. 100] of Integer;
```

ينشئ الـ array فى ذاكرة من ١٠٠ عدد صحيح . كـ array ديناميكى، يتم تعريف الـ IntArray أولاً دون نطاق من قيم الفهرس فى الاقواس :

```
var
 IntArray : array of Integer;
```

هـذا يشـيـر الى ان نوع الـ IntArray هو array من قـيم الـ Integer . لاستخدام الـ array يجب ان تقوم عبارة program بتخصيص مساحة للـ array لحمل هذه القيم.

والطريقة الثانية، والمستحبة، لإنشاء arrays ديناميكية هى باستدعاء الـ SetLength procedure، والذى كان يتم استخدامه فى الماضى لتغيير اطول الـstrings فقط. استخدم الـ SetLength لبدء array ديناميكى كمايلى:

```
begin
 SetLength(IntArray, 100);
   ...
end;
```

بعد تخصيص مساحة array، يمكنك استخدامه كما تفعل تماماً مع النوع الساكن. كما فى الـ strings، انك لا تحتاج الى تحرير المساحة المخصصة. ولكن، يمكنك إزالة مساحة ذاكرة الـ array الديناميكى بتحديد طوله بصفر بالعبارة:

```
begin
 SetLength(IntArray, 0);
   ...
end;
```

والاختيار الآخر هو ان تعين nil لمتغير الـ array يمكن تبسيط الـ code السابقة الى مايلى:

```
IntArray := nil;
```

يمكنك تعريف array ديناميكى من أى نوع. على سبيل المثال، هذه العبارة تعرف array ديناميكى من نوع الـ strings:

```
var
 StrArray : array of string;
```

يمكنك ايضاً تعريف arrays ديناميكية متعددة الابعاد، وهى حرفياً "arrays of arrays." على سبيل المثال، التحديد الساكن:

```
var
 Grid: array[1 .. 10, 1 .. 25] of Double;
```

ينشئ static array ثنائى الابعاد ذا عشرة صفوف و ٢٥ عمود من قيم الـ Double floating-point. كـ array ديناميكى، يمكن تعريف الـ array كمايلى:

```
var
 Grid: array of array of Double;
begin
 Grid := New(Grid, 10, 25);
   ...
end;
```

أو بدلاً من استخدام الـ New، يمكنك استخدام SetLength كمايلى:

```
SetLength(Grid, 10, 25);
```

وبأى الطريقتين، يتم تخصيص مساحة الـ Grid ثنائى الابعاد، ١٠×٢٥، لقيم Double floating-point:

وعلى عكس الـ arrays static، لا يجب ان يكون الـ arrays الديناميكية مستطيلة. بعبارة اخرى، ان صفوف واعمدة الـ arrays الثنائى الابعاد والأكثر من ذلك لا يجب ان تكون من نفس الحجم. بعد تحديد الطول أو استخدام الـ New كما تم التوضيح، يمكنك اعادة تخصيص أى صف أو عمود مع code مثل:

```
begin
 SetLength(Grid, 10, 25);
   SetLength(Grid[2], 12);
   ...
end;
```

ان العبارة الاولى تخصص للـ array Grid ١٠×٢٥ مستطيل من قيم الـ Double. والعبارة الثانية تعيد تخصيص الصف الثالث لهذا الـ array، مما يقللها

الى ١٢ قيمة. من الممكن ايضاً تخصيص الصفوف للـ arrays متعددة الابعاد فقط. على سبيل المثال، العبارة التالية تنشيء array، حيث تختلف الاعمدة فى الحجم من عنصر واحد الى أى عدد (n) من العناصر . اختبر هذا الـ code :

```
var
 Triangle: array of Integer;
    I: Integer;
begin
 SetLength(Triangle, 10); // Allocate 10 rows
    for I := 1 to 10 do
     SetLength(Triangle[I], I);
       ...
end;
```

بغض النظر عن كيفية تخصيص مساحة للـ arrays الديناميكية، فهى دائماً تكون مفهرسة من صفر الى arrays اصغر حجماً فى عدد العناصر . لضمان السلامة. استخدم الـ functions Low و High للحصول على الروابط القانونية للـ array الديناميكى. على سبيل المثال، فى هذه الجزئية :

```
var
 IntArray: array of Integer;
    I: Integer;
begin
 SetLength(IntArray, 100);
    for I := Low(IntArray) to High(IntArray) do
     IntArray[I] := I;
end;
```

للـ array الديناميكى، يقوم الـ Low(X) بإدخال صفر، لذا فالـ Loop السابقة يمكن كتابتها بسلامة كهذا :

```
for I := 0 to High(IntArray) do
...
```

وهذه تعتبر افضل طريقة دائماً لمنع خطأ المدى- وهو فهرسة عنصر array غير موجود بعد أعلى منطقة مخصصة. وتقوم الـ High function بارجاع عدد العناصر فى الـ array ناقص واحد.

Caution

تحذير: لا تقم بطرح واحد من الـ High(X). تقوم الـ function بارجاع عدداً يكون اقل من عدد العناصر فى الـ arrays الديناميكى X.

كن حذراً عند نسخ arrays ديناميكية- فهى لا تعمل كالـ strings. على سبيل المثال، اذا كان لديك arrays من نفس النوع والحجم تماماً، يمكنك تعين array واحد للآخر، كما هو موضح هنا:

```
var
 A, B: array of Double;
begin
 SetLength(A, 100);
   ... // Use array A
   B := A; // Copy A to B which must be equal types
   ... // Use arrays A & B
end;
```

الـ code يعرف إثنين من الـ arrays الديناميكية لقيم Double floating-point. الـ arrays الأول، A، يخصص مساحة فى الذاكرة، ويتم استخدامه. ان عبارة التعيين تنسخ الـ A arrays الى B، ولكن، لان A و B هى مجرد (pointers)، فإن العبارة تجعل A و B يخاطبا نفس الذاكرة. فهى لا تنشئ arrays منفصلين. ولكن، بخلاف الـ strings، فإن الـ arrays الديناميكية لا تستخدم الـ copy-on-write algorithms.

ان الـ Reference counting تحافظ على عدد الـ pointers address التى لها نفس الذاكرة. وفقط عندما يتم التخلص من الـ pointer النهائى يتم إدخال الذاكرة المخاطبة للمجموعة. ويؤكد الـ copy-on-write algorithms ان، اذا حدث تغيير لذاكرة تم مخاطبتها بواسطة اثنين أو أكثر الـpointers، نسخة جديدة من الذاكرة يتم عملها قبل السماح بهذا التغيير. وبذلك، اذا نسخت strings واحد

الى الآخر، فإن strings واحداً فقط يوجد فى الذاكرة حتى تقوم بالتغيير الى الـ strings المخاطبة. وهذه ليست الحالة مع الـ arrays الديناميكية. على سبيل المثال، فبعد عبارة التعيين فى المثال السابق، فهذا الـcode:

```
A[1] := 3.14159;
```

تقوم ايضاً بتغيير B[1] الى قيمة الـ pi.

يمكن تغيير احجام الـ array الديناميكى فى وقت التشغيل. فى الواقع، هذه واحدة من المزايا الاساسية للـ arrays الديناميكية. فهذه الـ code، مثلاً، تنشئ array من قيم الـ Double تستخدم الـ array، ثم تستدعى الـ Copy function لإعادة تحديد حجم الـ array:

```
var
 Values: array of Double;
begin
 SetLength(Values, 100);
   ... // Use array Values here
   Values := Copy(Values, 500);
   ... // Use larger array here
end;
```

فى هذا لـ code، الـ Values تعتبر array ديناميكى لقيم Double floating-point. ان العبارة الأولى فى البرنامج تعد الـ values لحمل مائة عنصر. والعبارة الثانية تستدعى الـ Copy function لإنشاء array اكبر يتكون من خمسمائة عنصر، وتنسخ القيم الموجودة للـ array الجديد. بعد ذلك، يتم ترك الـ array الاصلى.

Note

ملحوظة: اثناء الاستدعاء للـ Copy، يوجد arrays فى الذاكرة–واحداً من الحجم الاصلى وواحداً من الحجم الجديد. اذا احتجت الى توفير استخدام الذاكرة، فكن منتبهاً لهذا عند استدعاء الـ Copy لإعادة تخصيص مساحة الذاكرة.

وفى حالة الحاجة لـ arrays فعلية كثيرة من نفس النوع، يمكنك إنشاء نوع بيانات array ديناميكى باستخدام code مثل التالية:

```
type
 TIntArray = array of Integer;
    TFloatArray = array of Single;
var
 IntArray1: TIntArray;
    IntArray2: TIntArray;
    FloatArray1: TFloatArray;
begin
 SetLength(IntArray1, 100);
    SetLength(IntArray2, 200);
    SetLength(FloatArray1, 300);
    ...
end;
```

يتم تعريف نوعين من الـ array الديناميكى: TIntArray و TDoubleArray . يتم تحديد المتغيرات عندئذ باستخدام انواع البيانات والمساحة المخصصة باستخدام الـ SetLength. وميزة هذه التقنية هى سهولة التعديل لنوع الـ array. على سبيل المثال، فى الـ code السابقة، يمكنك تغيير نوع الـ TFloatArray هكذا:

```
type
 ...
    TFloatArray = array of Double;
```

هذا هو التغيير الوحيد المطلوب لتعديل كل الـ arrays الديناميكية الفعلية من نوع الـ TFloatArray من الـ single-precision الى الـ double-precision لعناصر الـ floating-point.

ملحوظة اخيرة:

هذا هو آخر فصل فى الكتاب، ولكن ليس آخر الموضوع. فى الواقع، ان خبرتك مع Delphi فى بدايتها. ان شعبية Delphi تزداد يوماً بعد يوم. فإن مطورى البرمجيات على مستوى العالم يتبعون Delphi ويكتشفون المزيد عن قيمة الـ objects-oriented objects اذا قرات هذا الكتاب، فإن لديك الآن قاعدة ثابتة فى تقنيات تطوير Delphi. اتمنى ان تكون قد استمتعت بالرحلة. والآن جاء دورك قم بالبحث فى online help، وقم بتعليم نفسك موضوعات جديدة بكتابة برامج اختبارية قصيرة.

How to Use the CD-ROM

The CD-ROM inside the back cover contains all sample projects described in this book. Copy the files from any CD-ROM directory to an empty directory on your hard drive and then open the copied project with Delphi's File|Open... or File|Open Project... commands. Browse to any directory and open the .dpr file-there is only one in most project directory paths. If you don't see a .dpr file, make sure the Open dialog's Files of Type box displays Delphi file (*.pas, *.dpr). Not all sample directories, however, contain projects. For these, there are no .dpr files to open.

Using the Listings

All listings in the book are located in named directories on the CD-ROM. None of these files is compressed. To use the files, copy any subdirectory from the Source directory to your hard drive. Or, if you have the room, copy the entire Source directory to a hard drive letter such as C:\ or D:\.



Some readers have requested that I organize sample applications by chapter number. This is a useful convention in some cases, but in Delphi 4 Bible, several different chapters refer to the same applications. For this reason, all sample applications are stored in directories named the same as their project names. For example, the SysColor demonstration's files are located in the Source\SysColor directory.

Although most directories contain full Delphi projects, some such as Misc contain .pas or other files. Use Delphi's File|
Open... command to view these files.

Running and Compiling Programs

All programs are compiled. If you just want to run them, use the Windows Explorer program to execute any .exe code file. You may execute most programs directly from the CD-ROM. However, those programs that write to disk files must first be copied to your hard drive or another writeable device.

To compile a program, you must first copy its directory to your hard drive. If you attempt to compile a program directly to the CD-ROM, Delphi will display an error message when it attempts to write to the .exe code file. If, after copying a directory to your hard drive you still have trouble compiling, the target files might be marked Read-Only. See "Unmarking Read-Only Files" next for more help.

Unmarking Read-Only Files

Copying CD-ROM files might set their Read-Only attribute flags, which will prevent writing to the .exe and .dcu code files. If, when compiling a program you receive an error that a file is in use or marked read-only, follow these steps to correct the problem:

1. Copy the Source directory from the CD-ROM to your hard drive directory C:\, D:\, or other.
2. Open a command-line prompt (MS-DOS) window.
3. Enter C: or D: (or another letter) to change to the Source drive.
4. Enter the command CD \Source to change to the Source directory.
5. Type the command ATTRIB -r *.* /S to turn off all Read-Only flags.
6. If you receive a "sharing violation" error in Step 4, exit Delphi and try Step 4 again.



الفهـــرس









****

رقـــم الإيـــداع
٩٩/١١٣٥٧ لسنة ١٩٩٩

الترقيم الدولـى
977-307-047-6







Write Dynamic Programs with Delphi

reate a wide variety of powerful applications for Windows 95, 98, and NT platforms! Providing step-by-step structions, expert tips, and over 100 sample applications, international bestselling author Tom Swan shows ou how to take full advantage of Delphi 4's capabilities to develop software in different categories, including raphics, animation, database management, the Internet, and more.

You Can Do It with Delphi 4

- Explore Delphi's environment, visual components, and forms
- Create sleek user interfaces by using toolbars, coolbars, and scroll panels
- Develop applications with the Windows clipboard, OLE, and DDE
- Master the techniques of Object Pascal string-handling, lists, file streams, and exceptions
- Generate application charts and reports with QuickReport and TeeChart component libraries
- Construct custom components and ActiveX controls
- Add the finishing touches to your application — debug, produce online help, and create dynamic link libraries

Delphi 4 running in Windows 95

The OleContainer application editing a bitmap file using Windows Paint

The Fishy application demonstrates Delphi's capability to read and write Blob fields in databases. In this case, the Blob is the bitmap image shown at right in the program's window

CD-ROM features source code from over 100 sample applications in the book!

About the Author

Tom Swan is the internationally acclaimed author of over 30 computer books, including IDG Books Worldwide's bestselling Type & Learn® C. He is a former contributing editor to PC Techniques and columnist for PC World and Dr. Dobb's Journal.





